
Universidade de Aveiro Departamento de Electrónica, Telecomunicações e Informática,
2009

José Ricardo Marques de Oliveira

Representação do Mundo para um Robô de Condução Autónoma
World Representation for an Autonomous Driving Robot

"All our knowledge begins with the senses, proceeds then to the understanding, and ends with reason. There is nothing higher than reason."

— Immanuel Kant

Universidade de Aveiro Departamento de Electrónica, Telecomunicações e Informática,
2009

José Ricardo Marques de Oliveira

Representação do Mundo para um Robô de Condução Autónoma
World Representation for an Autonomous Driving Robot

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática (M.I.E.C.T.), realizada sob a orientação científica do Prof. Dr. Artur José Carneiro Pereira e do Prof. Dr. José Luís Costa Pinto de Azevedo, Professores do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro.

o júri / the jury

presidente / president

Doutor Tomás António Mendes Oliveira e Silva
Professor Associado da Universidade de Aveiro

vogais / examiners committee

Doutor Agostinho Gil Teixeira Lopes
Investigador Auxiliar da Universidade do Minho

Doutor Artur José Carneiro Pereira
Professor Auxiliar da Universidade de Aveiro (Orientador)

Doutor José Luís Costa Pinto de Azevedo
Professor Auxiliar da Universidade de Aveiro (Co-orientador)

agradecimentos / acknowledgements

É com muito gosto que aproveito esta oportunidade para agradecer a todos os que me ajudaram durante a realização deste trabalho. Começo por agradecer ao meu orientador Prof. Artur Pereira, pelo apoio no trabalho realizado, pelas sugestões e correcções a esta dissertação. Assim como ao meu co-orientador Prof. José Luís Azevedo pelas correções sugeridas.

Desejo também agradecer aos colegas de equipa, Arturo Castillo e Miguel Sequeira, pelo excelente trabalho desenvolvido, bem como por todo o apoio prestado.

Finalmente, quero agradecer e dedicar este trabalho aos meus pais José Manuel e Idalina, irmã Isabel e à minha namorada Alice.

Palavras-chave Robótica, Representação do Mundo, Festival Nacional de Robótica, Condução Autónoma, Representação Métrica, Representação Topológica

Resumo

Condução autónoma constitui a deslocação de um agente, robô ou veículo, de um qualquer ponto no espaço para um outro, sem qualquer intervenção humana, por forma a atingir objectivos pré-estabelecidos.

Para conduzir de forma autónoma, usando planeamento de trajectória, é crucial que o agente consiga representar abstractamente tanto o conhecimento *a priori* acerca do mundo, como a informação que este vai adquirindo à medida que avança.

Para alcançar este propósito, desenvolveu-se um sistema para ser usado na pista da Competição de Condução Autónoma do Festival Nacional de Robótica. Este sistema caracteriza-se por ser flexível e modular. Tais caracteríticas permitem não só a adição componentes na pista acima referida, mas também a fácil expansão do suporte a outros tipos de pistas ou circuitos.

Concluiu-se, pois, que o modelo de representação mais adequado para o sistema que se pretendia desenvolver seria um modelo híbrido, na medida em que, ao nível global tal representação seria topológica e ao nível local métrica. Ou seja, dividindo a pista em secções, estas são a base para a representação topológica, sendo depois cada secção mapeada internamente de forma métrica.

Ao integrar o trabalho desta dissertação com o sistema global lograva-se alcançar um sistema de Condução Autónoma susceptível de planear a curto e médio prazo, com vista a melhorar o desempenho dos robôs usados no projecto, relativamente á solução anteriormente usada, que era baseada num sistema reactivo com alguma memória e noção de estado, mas sem planeamento de trajectória.

Keywords Robotics, World Representation, Portuguese Robotics Open, Autonomous Driving, Metrical Representation, Topological Representation

Abstract Autonomous driving is the movement of an agent, robot or vehicle, from some point in space to another one, without any human intervention, in order to achieve predetermined goals.

To drive autonomously using trajectory planning, it is vital to have an abstraction of the knowledge about the world, be it *a priori* or information that the agent acquires during the driving.

For this, we developed a system capable of abstractly represent, not only the track for the Autonomous Driving Competition of the Portuguese Robotics Open, but also, tracks with similar characteristics. The system was developed in a flexible and modular manner, in order to allow the addition of new elements to the stated track and the easy expansion to support other types of tracks and circuits.

The conclusion was that the most appropriate representation model for the system we were trying to develop was an hybrid model, in that, at a global level the representation would be topological and at a local level it would be metrical. In other words, dividing the track into sections, these are the basis for the topological representation, being each of the sections then mapped internally using a metrical representation.

Integrating the work of this dissertation in the global system, one hoped to achieve a Autonomous Driving system capable of short and medium term planning, with the goal of improve the performance of the ROTA project robots, comparatively with the previous solution, which was based in a reactive system with some memory and to some degree stateful.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Summary

This chapter starts with a brief reference to robotics, detailing, in particular, autonomous driving and some major projects in the latter field. After this, a small introduction to the Portuguese Robotics Open is presented, describing the different competitions, but focusing in the autonomous driving one. Then some details about the ROTA project are given, including a platform overview. Next the objectives that were proposed for the dissertation are enumerated. Finally the structure of the dissertation is detailed.

1.1 Robotics and autonomous driving

The first use of the word *Robotics* is attributed to Isaac Asimov, in his 1941 short story "Liar!" [1]. Robotics is seen as the science and technology related to robots [2]. It is a scientific area that involves various other fields such as: **Artificial Intelligence**, **Mathematics**, **Computing**, **Mechanics** and **Electronics**.

In what this work is concerned with, a robot is a piece of hardware, that has the ability to perform tasks in an autonomous manner. The use of robots to perform some repetitive tasks is common nowadays. As the time progresses, these autonomous machines are capable of executing more and more advanced tasks. Sometimes performing them cheaper or more reliably than humans. Robots are also used to execute possible dangerous tasks for humans. In sum, robots are used not only to achieve better results in production tasks, but also for replacing humans in dangerous tasks. In sum, they are used to increase the quality of life of the human race in general.

A particular field of research in robotics is **Autonomous Driving**. Autonomous driving has the goal of automate, fully or partially, some of the various driving tasks [3]. The development of autonomous driving capable robots is not only a research field. Some of the driving assist systems in today's vehicles are byproducts from the Autonomous Driving research. Autonomous vehicles are the next generation of driving systems. These will allow the reduction of road accidents and associated injuries, make a more efficient use of the roads, reduce the energetic cost for mobility and improve the mobility of goods and people [3].

The first documented autonomous robot dates back to 1977. The Tsukuba Mechanical Engineering Lab in Japan developed a robot with the purpose of following white lines similar to the ones normally painted in the roads. The robot achieved speeds of up to 30 km/h.

During the year of 1980, Ernst Dickmanns and his team re-engineered a Mercedes-Benz van (with the name **VaMoRs**) adding cameras and other sensors. Additionally, the steering wheel, throttle and brakes could be controlled by computer commands. This modifications allowed the modified van to achieve speeds up to 100 km/h, while driving autonomously.

After this achievement the European Commission started the **EUREKA Prometheus** Project (PROgraMme for a European Traffic of Highest Efficiency and Unprecedented Safety) with a funding of 800 million dollars. A number of universities and car manufacturers participated in this project, among these, was Dickmanns and his team at Bundeswehr University Munich.

In 1994 Dickmanns demonstrated a re-engineered S-Class Mercedes-Benz (**VaMP**), that drove more than 1000 kilometers on a Paris highway, with speeds up to 130 km/h, passing

slower cars in the left lane. Then in 1995, Dickmanns showed the **VaMoRs-P**, which was capable of driving 1600 kilometers in the German Autobahn, at speeds of up to 175 km/h, driving in traffic and executing maneuvers to pass other cars. Nevertheless, these cars still were semi-autonomous, with a human pilot present to ensure the safety of other drivers in the roads. The latter car, VaMoRs-P, had a mean distance of 9 km, with a maximum stretch of 158 km, without human intervention. The VaMoRs-P prototype overview is depicted in Figure 1.1.

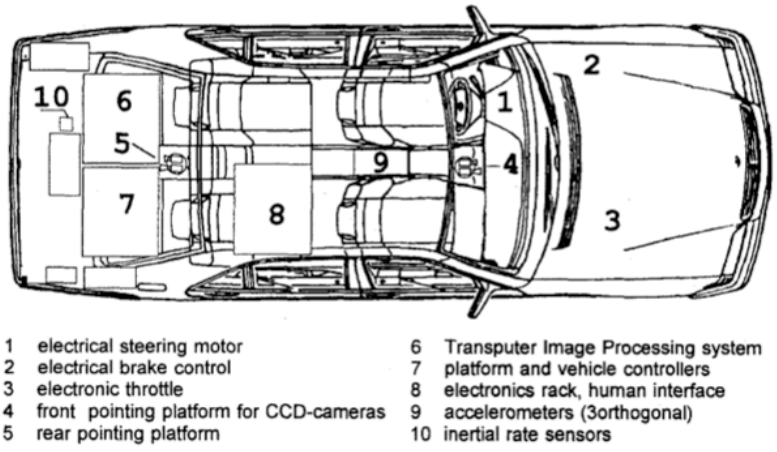


Figure 1.1: Top view of VaMoRs-P prototype, with the autonomous driving components depicted [4].

Another participant in the EUREKA Prometheus project was the italian **Artificial Vision and Intelligent Systems Laboratory (VisLab)** of the Department of Information Technology (Parma University). They developed a prototype called **MobLab**, in cooperation with the Polytechnic of Turin. This prototype was available to all the italian research units involved in Prometheus. After the project end, VisLab developed a number of different prototypes, such as: **ARGO**, **TerraMax** and **BRAiVE** (for more prototypes from VisLab see [5]).

ARGO was developed between 1997 and 2001. It was the first fully autonomous car developed by VisLab. This car was based on a Lancia Thema equipped with specialized devices to drive autonomously. In 1998 this car was used in the *MilleMiglia in Automatico* tour, to demonstrate that it was possible to automatically drive a vehicle, using only visual information and low-cost general purpose hardware. ARGO drove more than 2000 kilometers, with 94% of that distance being in fully autonomous mode. [6]

TerraMax is not one VisLab prototype, but a number of different vehicles used in the DARPA Grand and Urban Challenges. These DARPA (Defense Advanced Research Projects Agency) competitions are better described further on. **TerraMax 2004** completed 1.2 miles

of the 130+ miles of the 2004 competition. **TerraMax 2005** was one of the five teams that finished the 2005 course. **TerraMax T2** was used in the 2007 Urban Challenge, but did not finish the course. [5]

BRAiVE (BRAin-drIVE) is the latest prototype fully developed by VisLab and uses many of the systems created by them in the last 15 years. It is one of the best integrated vehicles capable of fully autonomous driving [7]. The amount of sensors and cameras used by this car can be seen in Figure 1.2.

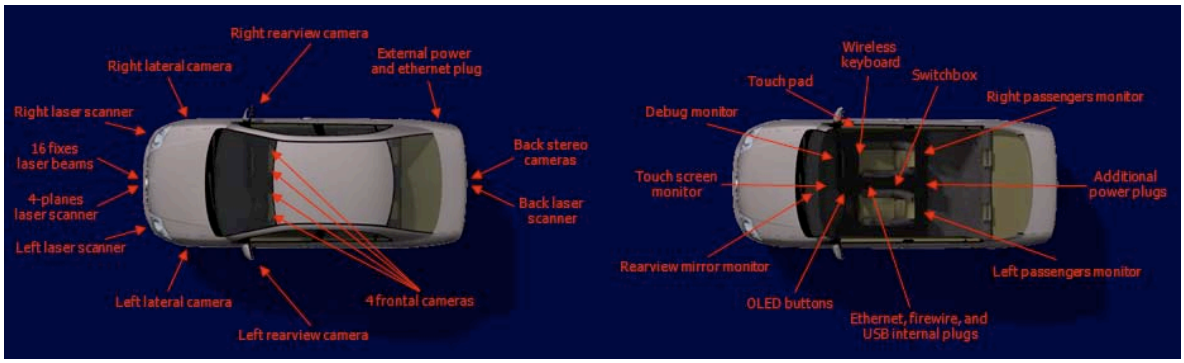


Figure 1.2: Overview of the BRAin-drIVE equipment [7].

On the other side of the Atlantic Ocean, in the United States, during 1980 the **Defense Advanced Research Projects Agency (DARPA)** funded the **Autonomous Land Vehicle (ALV)**. This vehicle, using laser radars, computerized vision and a robotic control mechanism, was able to drive autonomously at speeds of 30 km/h.

Since 1984, the **Carnegie Mellon University Navigation Laboratory** has built a series of computer-controlled vehicles capable of autonomous driving or driver assistance. Of note is the NavLab 5 vehicle, which, in 1995, was used in the *No Hands Across America* event. In this event, the NavLab 5 from CMU Navigation Laboratories drove a 3000 mile cross-country road, from Pittsburgh to San Diego, traveling autonomously 98.2% of the way. Despite this, the vehicle used was only semi-autonomous, as only the steering wheel was controlled in an automatic way, being the throttle and brakes human operated.

More recently, we should highlight an event, promoted by DARPA, the **DARPA Grand Challenge**. It was a prize competition, in the field of autonomous driving, in the years of 2004 and 2005. The 2004 competition was a 228.5 kilometers course in the Mojave desert, but none of the competitors even finishing, with the best mark being 11.84 kilometers. In 2005 a total of 5 teams completed the course, but all except one team reached the 11.84 kilometers mark from the previous year.

In 2007 a similar competition took place named the **DARPA Urban Challenge**. This

competition takes place in an urban environment, in a 96 kilometers urban area course. The vehicles need to comply to traffic regulations and, at the same time, avoid possible obstacles (this includes other team's vehicles). While in the Grand Challenge the vehicles operated isolated from each other, in this competition the vehicles share the same urban course. Six teams successfully finished the course.

1.2 Portuguese Robotics Open

The annual **Portuguese Robotics Open (Festival Nacional de Robótica - FNR)**, which is held since 2001, is a scientific event promoted by the **Sociedade Portuguesa de Robótica**, with the main goal of promoting Science and Technology among students of all scholar levels, as well as among the general public, using robot competitions. Besides the competitions, robot and robotic systems demonstrations take place as well as a Scientific Conference, in which students and researchers can present their latest work in the field of robotics [8]. This event has seen a substantial growth, both in the number of participants, and also in the audience. The main competitions in the FNR are divided in two levels, **Senior** and **Junior** competitions.

The senior competions are [8]:

- **Autonomous Driving** - *"Represents a medium complexity technical challenge, in which an autonomous mobile robot must travel a path along a closed track, which has striking similarities with the driving of a vehicle on a conventional road. The track is bordered by two white lines, has two lanes separated by a dashed line, the approximate shape of an 8, an intersection in the center with a crosswalk and a pair of signal panels, and a tunnel on one of the curves. The position of the obstacle on the track, the exact location of the parking area and free parking slot in this area are unknown to the robot at the start of his trial."* [9] For more information about this competition, see Section 1.2.1.

- **Middle Size League** - *"The Middle Size League (MSL) is an official RoboCup league. Two teams with (typically) 5 robots each, that can have up to 80 cm height, 50 cm in diameter and 40 Kg of weight, challenge each other in a football field . This field is similar to the human one which has 11 players, but with a smaller size (18m x 12m). [...] The robots can have numerous sensors (cameras, gyroscopes, sonars, etc.) and actuators (motors, kicking devices, etc) which allow them to play with complete autonomy, i.e., without human intervention. The robots also have wireless connection to communicate*

with each other and the referee. Some elements have specific colors (the ball is orange, the field is green, the lines and goals are white) in order to allow an easier detection by the robots. There are usually two referees, one that regulates the game, and another which interacts with a graphical interface. This interface communicates to the robots the faults, goals, cards, etc, which occur during the game." [8]

The junior competitions include [8]:

- **Junior Search and Rescue** - *"In this competition mobile robots are used to identify victims quickly and accurately in disaster scenarios that are recreated artificially. These scenarios will range in complexity from line-following on a flat surface through paths with obstacles, line breaks and slopes, to reach an area where the victims are randomly placed in open terrain."* [8]

- **Junior Dance** - *"[...] consists in the realization of a choreography in which one or more robots "dance" to the rhythm of music, being evaluated by a panel of experts in Robotics and Dance. From the point of view of programming, this competition is very little demanding. Nevertheless, the final result, which is the combination of the movement of the robots with the music along with the imagination that is put on some choreographies, achieves good levels of artistic beauty."* [8]

- **Junior Soccer** - *"This competition is based on a two by two fully autonomous robots, filled with sensors and who's limit dimensions are 22cm, playing soccer. An infrared emitting ball and two different sized soccer fields, with different complexity in the programming level, are the remaining subjects for this exciting competition filled with soccer strategies and goals. The soil of the fields is green and the goals are colored blue and yellow so that robots may identify them. On the simplest field version there are no sidelines or goal lines and it's allowed to play with the protection walls. The second filed version have sidelines and goal lines and ball is not allowed to leave from inside those lines."* [8]

1.2.1 Autonomous driving competition

The autonomous driving competition of FNR had its first edition in 2001, the first year in which FNR took place. Its main goal is to promote technical developments in systems and techniques used in the field of autonomous driving. Although nowadays the competition is held in an indoor structured area, in the future some other spaces will be used such as parks or roads [10]. In this competition the track is delimited with two white lines, which simulates

a two-way road. Additionally, there are a number of different elements in the track, in order to increase the resemblance to a real world environment: a crosswalk with traffic light panels, one for each lane, a tunnel, a roadwork area, one or two obstacles and a parking area, being the latter two in an unknown position.

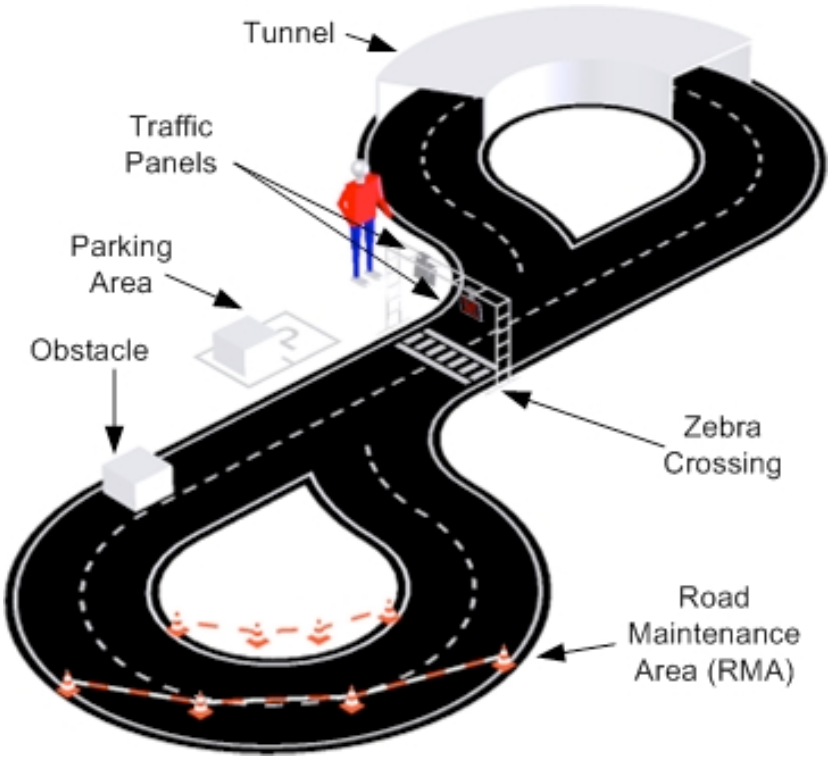


Figure 1.3: Overview of the Festival Nacional de Robótica autonomous driving track, used since 2007 [11].

The competition unfolds into three stages, that take place in three consecutive days, with increasing levels of difficulty from stage to stage. In each stage the robot can do 4 trials with a time limit of 10 minutes per stage. In all stages the robots start from the crosswalk and, after recognizing a green sign in the traffic lights panel, complete two laps around the track. The main goal is to take the least amount of time to complete each trial, *i.e.*, the robot that finishes the three stages in the least time wins the competition. The specifics for the stages are as follows [12]:

- **First Day** - in this stage each robot only needs to complete two full laps around the track. No obstacle, roadwork or tunnel will be placed in the track, the traffic lights panels are only used to trigger the start of each trial.
- **Second Day** - in this stage the traffic lights panel will give directions using 5 different traffic signs, which the robot will need to obey. One obstacle will be placed in the track

in an unknown position, which the robot may need to avoid, in case of possible collision. But no tunnel nor roadwork will be used. At the end of each trial the robot must park itself in the parking zone.

- **Third Day** - in this stage the signaling panels will be fully operational, as in the second day, but instead of one obstacle in the track there will be two obstacles placed in the track. A tunnel and a roadwork zone will also be placed in the track. At the end of each trial the robot must park itself in the parking zone, but one of the parking slots is occupied by an obstacle, so the robot needs to locate the free slot and park there.

1.3 The ROTA platform

Since 2001, the first year in which FNR took place, the **Electronics, Telecommunications and Informatics Department (DETI)** has attended 8 editions of the Autonomous Driving competition. During this period, various robot platforms have been developed, namely: **Cyclop** in 2001, **CaPicUA** in 2002, **CharrUA** in 2003, **RobIEETA** in 2005, **ROTA** in 2006, **RatoZinger** in 2008. In terms of classification DETI scores 3 first places, 1 second, and 2 thirds.

The latter two cars, depicted in Figure 1.4, are the ones used in the ROTA project. These are the last generation of autonomous cars in the history of DETI, used for development, demonstration and participation in competitions, such as the FNR Autonomous Driving.

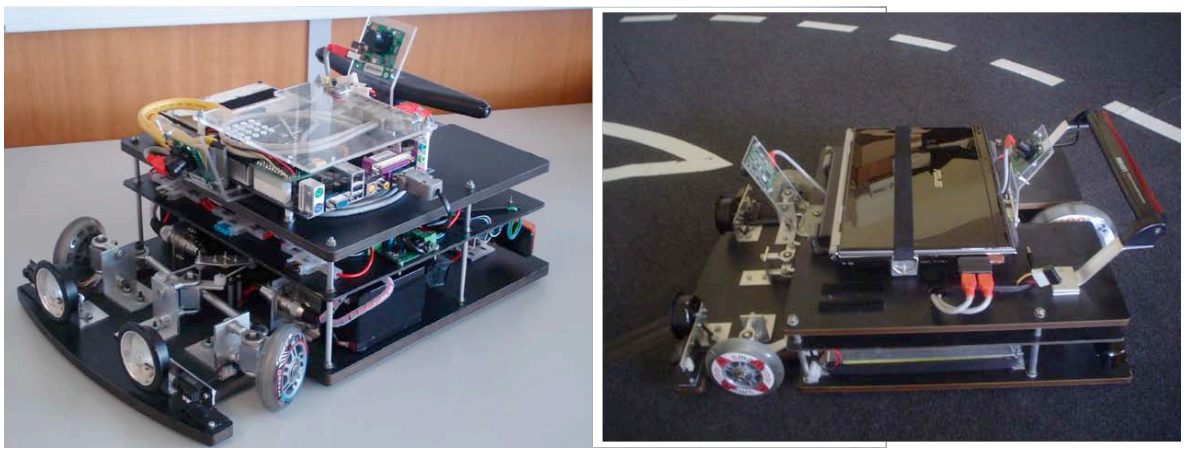


Figure 1.4: On the left the ROTA robot [10], on the right RatoZinger.

The ROTA robot is built upon a rectangular chassis, which supports the traction motor, the batteries and the steering controls. The remaining parts are placed in levels above this chassis. The first level above the chassis supports the electronic modules, such as the motor

controller and the interfaces to the sensors and actuators. The upper level contains the PC and the vision system composed of two standard FireWire cameras [10].

The ROTA platform can be subdivided in four major components, namely: high level control, low level layer, vision layer and mechanical layer. These components and their relations are depicted in Figure 1.5.

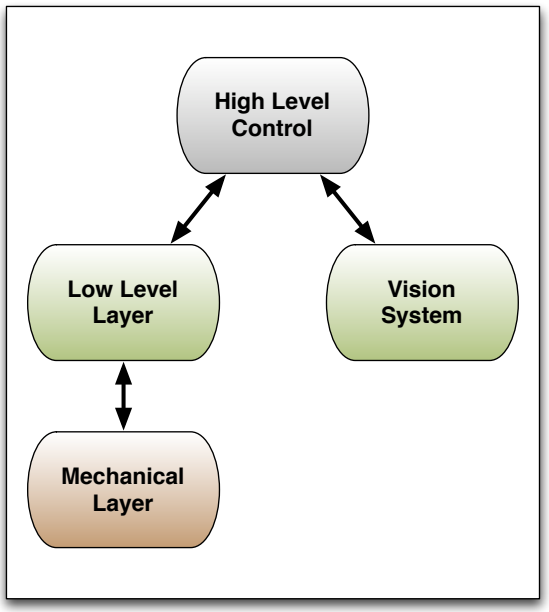


Figure 1.5: ROTA platform.

The **mechanical layer** is a tricycle with an Ackermann direction system and rear wheel drive, powered by two independent batteries. Each of the batteries powers a different part of the car. One supplies the logic blocks of the system, which include the high-level control and the low-level layer micro controllers, while the other supplies the traction motor. Using two independent batteries improves the operation reliability and is very useful during development, as we can start only the control logic and, just by pushing the car, analyze its behavior [10].

The **low level layer** (see Figure 1.6) is composed of a set of nodes interconnected by means of a CAN[1] network. A gateway interconnects the CAN network to the PC at the high level control using either a serial or USB port. This layer is responsible for: motion control, system monitoring and interface to other sensors/actuators [10].

The **vision system** for both robots is composed of two IEEE1394 cameras. Both of them are directed towards the front of the robot, but while one of these cameras is facing downward,

[1] *Controller Area Network*

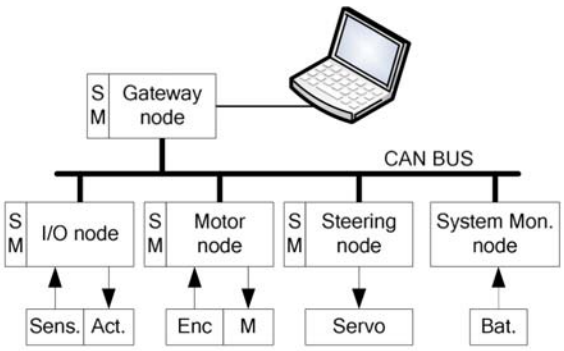


Figure 1.6: ROTA low level hardware architecture [10].

which is used for navigation purposes, the other camera is faced upwards and is used to sense the signaling panels information [10].

The **high level control** layer implements the global control for the robot. It collects data from the cameras and from the low-level sensing system and sends commands to the motion controls. In the ROTA robot this was composed of a mini-ITX board running a Linux operating system. This solution was selected due, not only, to the small form factor of the mini-ITX boards, but also, to the board technical specifications [10, 3].

Nevertheless the fact that RatoZinger is based in the ROTA robot, there are some fundamental differences between them. Some differences are mainly related to shortcomings found in the ROTA robot. The testing and debugging in the ROTA robot was difficult, being this related to the lack of a screen in the mini-ITX solution used in the robot. So, in the newer RatoZinger robot instead of using a mini-ITX, a standard laptop is used. This fact resolved the problem with the lack of a screen for testing purposes. But it also brought an additional benefit, laptops come packaged with built-in batteries, this increasing the longevity of the robot batteries. But, in order to support the new high-level control element, which is larger than a mini-ITX board, the physical dimensions of the RatoZinger robot are bigger than those of ROTA. Another shortcoming with the ROTA robot was the low distance to the ground of the chassis, this making the robot hit some small elevations in the track. To solve this, RatoZinger chassis has a higher distance to the ground.

Other differences are directly related to the advances in technology. For instance, the RatoZinger batteries are much lighter, thinner and smaller than the ones used in the ROTA robot. But, despite the smaller size, the new batteries provide increased capacity with a longer time between charges. Also the traction motor used in RatoZinger can achieve higher maximum speeds than the one used in the ROTA robot.

1.4 Objectives

The main goal for the ROTA project this year was to prepare the robots to drive autonomously using trajectory planning in the FNR 2009 edition. In order to achieve this, the almost purely reactive control scheme used in the past FNR editions was abandoned, migrating to a more deliberative control scheme. The idea is to do trajectory planning based on a proper representation of the track and of the elements contained in the track. The world representation used should not be limited to the current year FNR track, but be abstract enough to support future changes and also similar tracks.

So, the basic major tasks in the ROTA project were established, as follows:

- Create a proper world representation, that can be easily used by the other modules in the system.
- Enhance the perception methods currently used in the ROTA project, in order to update the world representation.
- Create a planning component, that can, based on the information contained in the world representation, obtain a motion plan for the robot.
- Devise a control mechanism capable of performing motion plans supplied to him.

More specifically, the main work for this dissertation, is related with the first element of that list, namely, to implement a world representation that could support the new trajectory planning system planned for use in the 2009 ROTA project.

As such, some major objectives for the world representation system, include:

- Use a flexible and expansible method for the world representation. Expansible because future changes to the FNR autonomous driving competition track can be easily mapped in the representation. And flexible to allow the use of the representation system in tracks with similar characteristics.
- Provide a stable and verified external interface. So that other modules in the system can easily add, update and access the world representation, without knowledge of the internal structure.
- Make use of available tools and features for rapid development and easy maintenance.

1.5 Dissertation structure

In order to enhance the clarity of this document, the remaining information here contained is organized in four chapters, in addition to the present introductory chapter, as follows:

- **Chapter 2 - Autonomous Robotics -** In this chapter some references about autonomous robotics are presented, starting with the basic control schemes for autonomous agents. Because some of this control schemes need an abstract world representation, some types of world representation are also introduced. In this chapter two agent architectures - BDI and Triple Tower - are briefly described as, these architectures have a direct relation with the architecture model of the ROTA project.

- **Chapter 3 - Architecture Model -** In this chapter the general architecture model for the ROTA project is discussed. The main subject of this dissertation - Track Representation - is also put into perspective, establishing a relation between it and the other entities of the system.

- **Chapter 4 - Track Representation -** In this chapter the main subject of this dissertation is detailed and discussed - the Track Representation used in the ROTA project. The use of an Hybrid Topological-Metrical world representation with heavy use of C++ features such as inheritance, polymorphism, overloading and templates, is also detailed.

- **Chapter 5 - Conclusions and future work-** The final chapter concludes the dissertation. It starts with a quick review of the work and details the conclusions to which we came during the planning and development phases. In this chapter some possibilities for future work directions are also given.

Chapter 2

Autonomous Robotics

Summary

This chapter focus in previous work done in the area of trajectory and motion planning. Starting with reactive control, used in previous generations of the ROTA project, passing through the more advanced deliberative control and ending in the hybrid control scheme. The deliberative and hybrid control schemes expose the need to abstract the world knowledge. This way some representation models are detailed. Finally, a brief introduction to some AI architectures, with interest for the ROTA project, is given.

2.1 Control

Agent control can be seen as the mechanism which, using information about the environment where the agent is inserted, takes decisions to make the agent progress and achieve its goals. There are many control schemes and in this section three of them are going to be detailed, because these are the ones with a direct relationship with the ROTA project, namely:

- Reactive control
- Deliberative control
- Hybrid control

2.1.1 Reactive control

Maja J. Matarić *et al.*, states the following about **reactive control** in [2]:

"Reactive control is a technique for tightly coupling sensory inputs and effector outputs, typically involving no intervening reasoning to allow the agent to respond very quickly to changing and unstructured environments."

In the reactive control scheme, the reasoning involved is very simple or not present at all. The agent sensory input data is used by the system in order to quickly evaluate the world state, and react accordingly. Because of this, a reactive control scheme does not require a "complete" world representation. Although a collection of sensory data can be considered a world representation, it has no past information associated with it, it is only the current state of world from the agent perspective. However, the past and present information, after some degree of processing, constitute a complete world representation.

Reactive systems have rapid responses to changes in the environment. These reflexive responses are embedded in a collection of preprogrammed, concurrent **behaviors** with minimal internal state (see Figure 2.1). Which are especially well suited for a dynamic and unstructured environment where the delay of having to access and interpret a complete world representation would be unrealistic [2]. These behaviors are no more than a set of rules that represent actions to perform when some conditions are verified, see Table 2.1 for some basic examples.

Because of this, these systems can have a very simple implementation, and if used in a correct problem can be very effective. However it should be noted that the pure reactivity includes the inability to store internal representations of the world, as been said in a previous

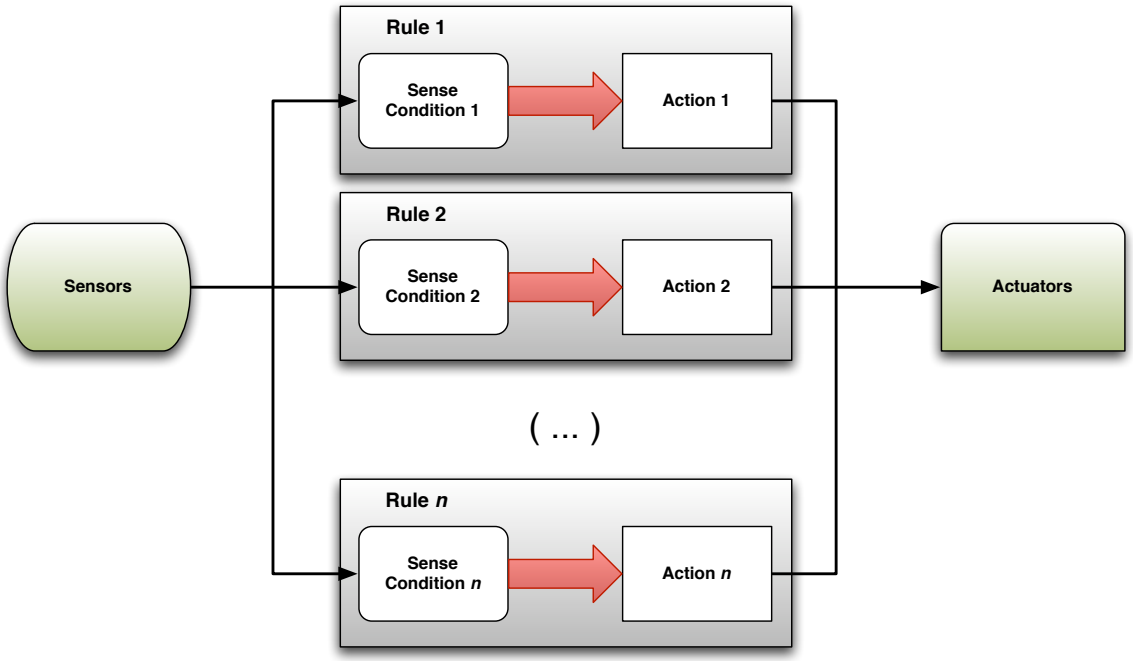


Figure 2.1: Behavior based reactive control scheme - there is no reasoning involved.

Conditions	**Actions**
if wall in front	*go back*
if no obstacle	*wander*
if opening on the left	*turn left*
if traffic light red	*stop*

Table 2.1: Behaviors and the condition-action tuple.

paragraph. Making these systems incapable of learning, improving over time and planning ahead in time. In sum, reactive systems trade off complexity of reasoning for a fast reaction time [2].

2.1.2 Deliberative control

Maja J. Matarić *et al.*, states the following about **deliberative control** in [2]:

"In this kind of control scheme, the agent uses all of the available sensory information, and all of the internally stored knowledge, to reason about what actions to take next."

Instead of the reflexive responses characteristic of the reactive control scheme, in deliberative systems a variable degree of reasoning/decision is involved, as such this kind of control scheme behaves more like an "intelligent" agent. Reasoning is typically in the form of planning, where the system checks all the available courses of action and each of their results, and only then decides which is the more efficient/fast, *i.e.* the decision which best suits the needs of the agent. Planning is known to be a computationally complex process, that involves combining all the sensory data available to the agent with all possible previous knowledge about the environment the agent is in, producing a plan that can be executed by the agent actuators [2].

Planning requires the existence of an internal, abstract representation of the world, which allows the agent to predict the outcomes of various actions in different states, in order to generate plans. Because of this, the internal world representation must be kept accurate, and up to date, in order to guarantee that the agent predictions have a close relationship with the actual actions and their outcomes. If this latter condition is met and with sufficient computing time to generate a plan. This allows the agent to act in a strategic manner, selecting the best course of action for any given situation [2]. This would be perfect if the normal environment where the agent is inserted was a perfectly structured and constant/static environment, which rarely happens. In some domains this approach is simply not feasible. See Figure 2.2 for a graphical representation of the deliberative control scheme.

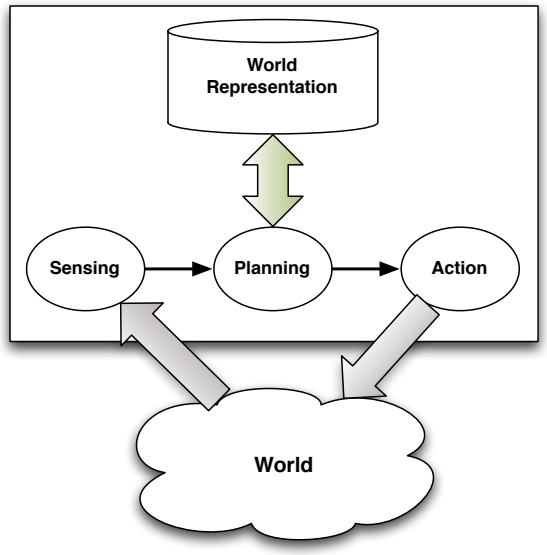


Figure 2.2: Deliberative control scheme.

2.1.3 Hybrid control

Maja J. Matarić *et al.*, state the following about **hybrid control** in [2]:

"Hybrid control aims to combine the best aspects of reactive and deliberative control: the real-time response of reactivity and the rationality and optimality of deliberation."

In this kind of control scheme there are two different components [2]. One related with the reactive behaviors, that deals with the agent immediate needs. The other component related with the deliberative decisions, that uses the abstract world representation to develop long term strategies, *e.g.*, long-term path planning. Both of these components need to interact with each other in order to mutually benefit themselves. This way, the reactive part has a higher priority in the case of an unexpected situation, but this component also has to realize the existence of the deliberative part, in order to take advantage of the long term strategy and planning that the latter one can provide, in order to obtain a more efficient and optimal result [2].

Hybrid systems normally use a *three-layer architecture*. The higher layers have increased intelligence but have lesser control [2] (see Figure 2.3). Being the layers as follows:

- Reactive layer, responsible for execution
- Deliberative layer, responsible for decision/planning
- Intermediate layer, responsible for iteration between the reactive and deliberative layers

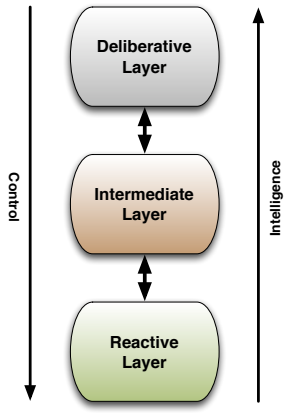


Figure 2.3: Three-layer architecture.

2.2 World Representation

It was established in the previous sections that both the deliberative and hybrid control schemes need to have a correct, up-to-date abstract world representation, in order to decide the correct course of action in any situation. This world representation must include both the robot and the environment where the robot moves. In autonomous driving the environment is typically a track and all the associated elements, like traffic lights, roadwork areas, delimiting lines, among others. In this section three types of world representation are detailed:

- Metric representation
- Topological representation
- Hybrid Metric-Topological representation

2.2.1 Metric representation

Metric representation uses the geometric relationship between objects/features in order to map them [13], *e.g.*, map a line in a 2D spatial referential, using the relationship between the center of the referential and the line. The metric representation can be supported in various ways; two of them are [13]:

- Cartesian coordinate system - uniquely specifies a point in 2D space using only a pair of values.
- Occupation grid map - a grid map is created and the state of each grid cell is marked as occupied or free.

The main advantage to this type of representation is the very precise environment mapping that can be achieved. But, on the other hand, this can increase the complexity in processing the map data, and can also increase the space needed to store the map itself [13].

2.2.2 Topological representation

Topological representation uses **landmarks** to map the environment. Landmarks are one or more distinctive features on an object or locale of interest. A landmark does not need to be a single object, it can be a grouping of obstacles [14]. In terms of representation, topological maps can be well represented by a graph [15], where the nodes represent relevant features or places and the edges represent the connections between them [13], see Figure 2.4.

Topological representation is more suited to represent larger environments than metric representation, because of the less space needed to store the map and also because of the easier planning allowed by the graph structures.

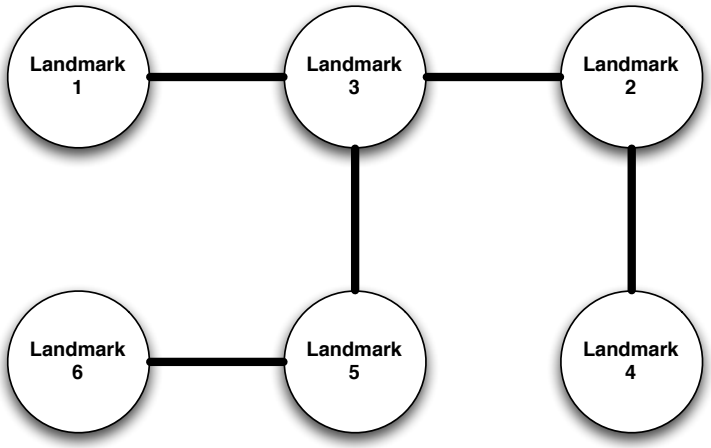


Figure 2.4: Topological landmarks.

2.2.3 Hybrid Metric-Topological representation

In an hybrid system both types of representations are used to map the world. The specific hybrid model that this work focus on uses a global topological map with local metric maps [15]. In this model the environment is represented using two levels:

- A higher level topological representation
- And a lower level metric representation for each of the nodes defined in the higher level one

This way the robot navigates metrically until it finds a distinctive place (landmark). At this time, it uses the topological representation to navigate to another node. Finally, when inside the new node, it returns to the metrical representation for navigation.

2.3 AI Architectures

Choosing the way to represent the world is only part of the problem. In order to build the high level control software of an autonomous robot, using an approach which includes deliberation, an architectural model is required. In this section a couple of general purpose architectures are detailed, that were used as a basis for the ROTA architectural model:

- Beliefs Desires Intentions architecture
- Triple Tower architecture

2.3.1 Belief-Desire-Intention

The Belief-Desire-Intentions (**BDI**), is a software model for AI agents, that uses three basic entities: **Beliefs**, **Desires** and **Intentions**. Paolo Busetta *et al.*, in [16], defines these basic components of the BDI architecture as follows:

- **Beliefs** - "represent the informational state of the agent, that is, what it knows about itself and the world", this entity represents the knowledge that the agent has about the world.
- **Desires** - "are its motivational state, that is, what the agent is trying to achieve", these are the final goals the agent expects to achieve.
- **Intentions** - "represent the deliberative state of the agent, that is, which plans the agent has chosen for eventual execution", are the steps that the agent has chosen to take, based on the current Beliefs, in order to achieve the its desires.

2.3.2 Triple tower

The Triple Tower architecture defines three different entities. Each of these entities being represented by a "tower", being each tower capable of interaction with the other towers, as can be seen in Figure 2.5.

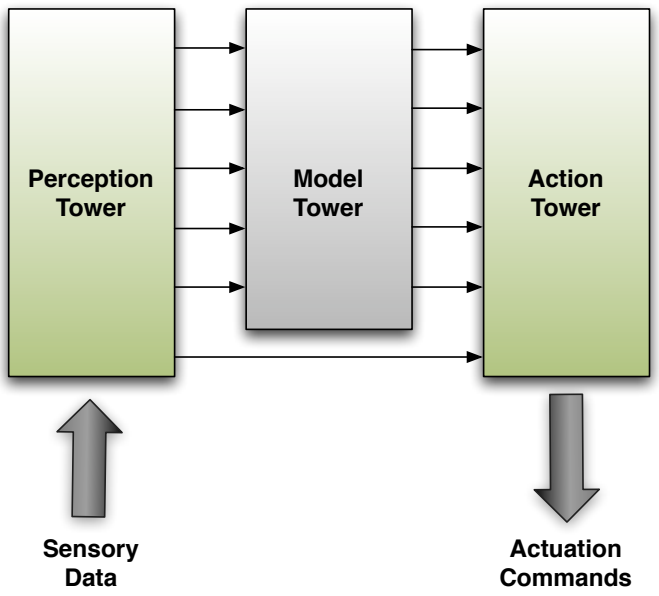


Figure 2.5: Triple tower architecture [17].

The leftmost tower, **Perception**, is the responsible for the sensory data processing. It starts with the raw data and processes this information into a more abstract representation [17]. The intermediate tower, **Model**, not only stores, but also ensures the integrity of the abstract world representation. The data captured and processed by the perception tower is used to maintain the model [17]. The rightmost tower, **Action**, using the information in the model tower, is responsible for the planning and execution of the agent goals. The connection between the perception tower and the action tower, reflects the reactive capabilities of the system, *i.e.*, the action tower can interact directly with the perception tower [17].

Chapter 3

Architecture Model

Summary

This chapter introduces the Architecture in which the motion planning robot is based, and also draws relations with some general purpose architectures, such as the Triple Tower and BDI models, described in the previous chapter, that are related with the model chosen for the ROTA robot. Other parts of the architecture, besides the supporting structures, are also detailed.

Regarding the FNR competition scenario, the world can be described has a track with straight and curve sections, a crosswalk, a tunnel and some traffic lights. Additionally to these basic elements, others can appear: one or two obstacles in unknown positions, a roadwork area that can redirect the "traffic" off the track and, a parking lot with two slots. One of these parking slots can already be occupied.

The environment in which the car navigates is semi-structured and dynamic. It is structured, because the lines and road patterns are well defined along the track. At the same time, it's not structured, because the roadwork area shape and position is unknown in advance by the car. It is also dynamic because, both internal and external elements to the track can change. This can bring significant changes to the perception of the world or even to its representation. Some examples of internal elements include: an obstacle that is placed in the track or the traffic lights panel changing the signal displayed. One case of an external element to the track is the changing light conditions altering the perception the agent has about its surroundings.

These conditions reveal the need for a dynamic world representation, that can adjust as the track and its elements change. The model should be abstract enough in order to support additional elements added to the FNR track and also to be used in tracks similar to the one used in that specific competition. Thus, creating a model that is independent from the FNR track. Because the basic structure of the FNR autonomous driving competition track is a fixed one (8-shaped track), the model must provide mechanisms to load the *a priori* knowledge about the world into the representation.

3.1 Architecture Overview

As discussed in the Objectives section, the main goal for the ROTA project is to add autonomous driving capabilities based on trajectory planning. Additionally to this goal, some additional ones were defined: enhance the perception methods used and provide a proper world representation. Considering these goals and the intention to define a modular structure that could easily allow:

- the addition of new perception mechanisms, with minimal changes to the rest of the system,
- the changing of the planning model, without the need to adapt anything else,

an architecture for use in the ROTA system was defined.

This architecture should have some main concerns, which are related basically with the track and the way the system interacts with it. Those concerns are listed below:

- Good understanding of track morphology and characteristics.
- Ability to dynamically acquire new features about the track characteristics, such as obstacle and roadwork insertion.
- Ability to correctly locate the car in the track.
- Ability to plan and execute long-term trajectories, calculated dynamically.

Thereby, the global ROTA architecture can be seen in Figure 3.1.

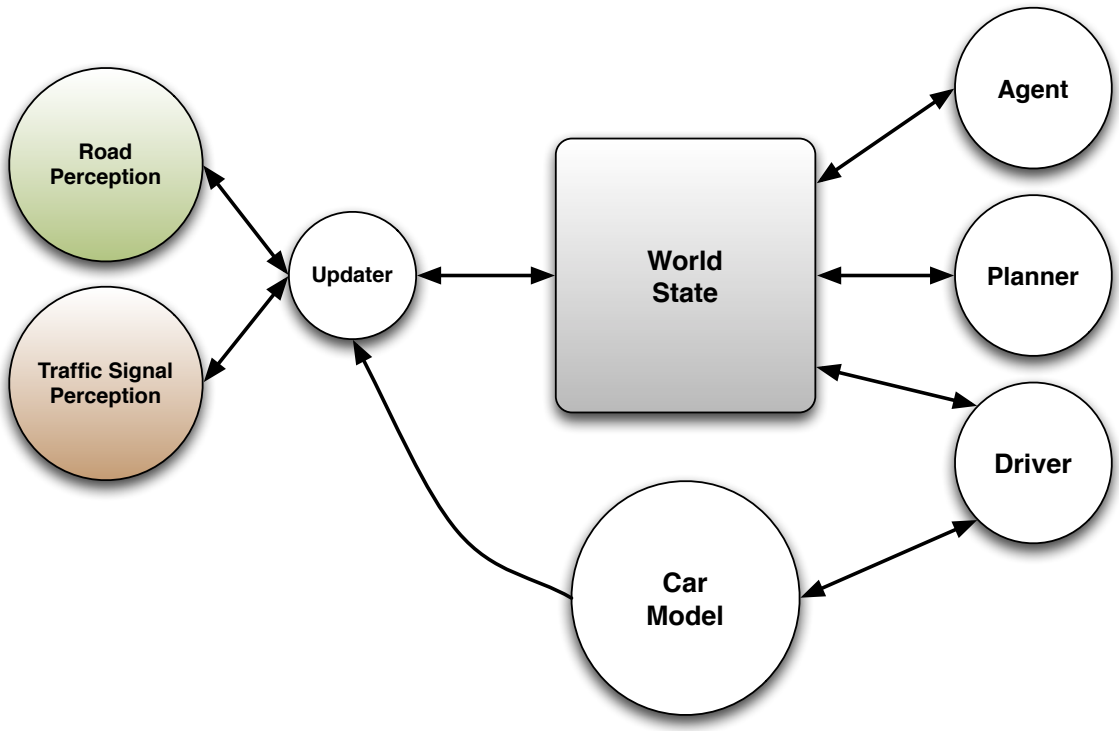


Figure 3.1: The ROTA model overview.

This is a data-driven approach, being the data stored in a shared structure accessible by the various processing modules. Although not mandatory, these processing modules, may be implemented through independent threads of execution. The state of the world includes the track representation, including its elements, the state of the car, and any other relevant data. The track is clearly the most complex and important piece among all the elements that compose the world state. It will have an in-depth study in the next chapter.

The Agent module represents the high level control, responsible for keeping the other modules under control, coordinating the whole system. The system launch and setup are the

responsibility of this module. It initializes all the needed structures and components, in order to make the system function properly. All the other modules in the system will be discussed further on, in this chapter.

The proposed architecture model for the ROTA project can be directly related with some general purpose models, in which it was loosely based, namely: the **Triple Tower** and **BDI** models.

When modeling the ROTA system with the Triple Tower architecture (see Section 2.3.2 for details about that architecture), the following organization is achieved (see Figure 3.2):

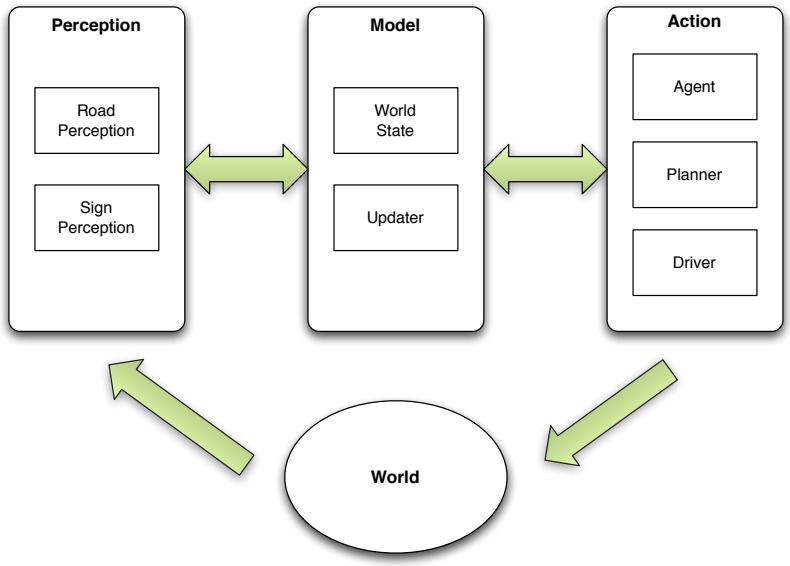


Figure 3.2: The ROTA system modeled using the Triple Tower architecture.

As in the Triple Tower architecture, the ROTA model can be divided in three levels (Towers). Each of these levels are detailed in the next paragraphs.

The **Perception *Tower*** contains all the high level perception mechanisms used in the system. In Figure 3.2 these two distinct mechanisms are shown. In the future additional ones can be added to the system without changing the base architecture. This tower is responsible for obtaining sensory input data and then interpret it. This data is then supplied to the model tower, in order to be integrated in the world representation.

The **Model *Tower*** contains the abstract representation of the World, which in this case, is mainly the track features and morphology. It also provides a high-level abstraction to some of the actions that can be performed to the track/world. This level receives the data from different sensing mechanisms in the perception tower, integrating the information and

updating the world state accordingly.

The **Action *Tower*** is responsible for the motion planning, as calculated by the Motion Planner module. And for the subsequent plan execution, by the Driver module.

In addition to the triple tower architecture, the ROTA system can also by modeled in the **B**eliefs, **D**esires and **I**ntentions architecture (see Section 2.3.1 for details about that architecture), as can be seen in figure 3.3.

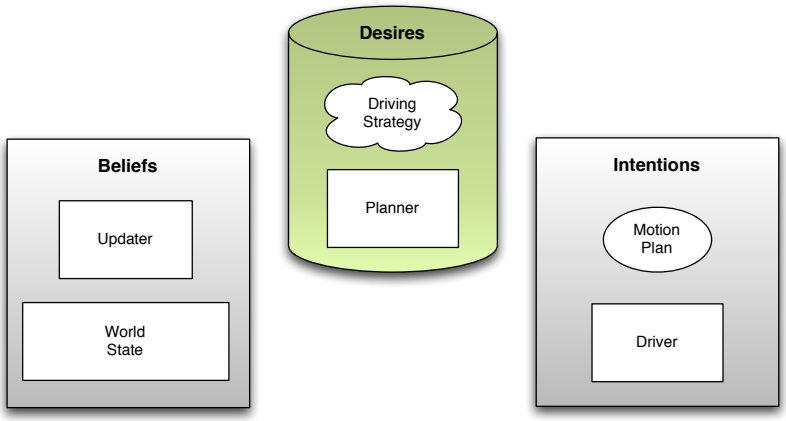


Figure 3.3: The ROTA system modeled using the BDI architecture.

Based on the **Beliefs, Desires and Intentions** architecture the system can de divided in the following parts:

- **Beliefs -** includes the World State, in this case, the Track representation and information about the Car and it's current state.
- **Desires -** this is composed by the Planner, and by the driving strategy - drive left, drive right, fastest trajectory, etc.
- **Intentions -** this is composed by the Motion Plan, calculated by the Planner module. And by the executor of the plan - Driver.

In the next sections a more in-depth analysis of each of the main components of the architecture model will be presented.

3.2 Car state and model

3.2.1 Car Model

As described in the introduction, the two vehicles in the ROTA project have a tricycle shape with a single traction wheel on the rear and two steering wheels in the front. This

structure allows the vehicle to move in a straight or curved line, being the minimum curvature radius of about 1 meter.

Actuation in the car locomotion is based on two parameters, one being the velocity to be applied to the rear wheel and the other a set point to the steering position that makes the car move in a given direction.

However an abstract model for the car was defined. It hides the constructive features of the car and provides a mathematical model to the other modules of the system. This abstract car model is supplied, by the higher level modules, with a desired curvature and velocity. Using the implicit knowledge about the car, the model determines the correct velocity for the traction motor and the correct set point for the steering position, in order to achieve the requested objectives. Returning a measurement of the distance traveled by the car.

This abstract model is implemented by the **CarModel** class. In addition to provide a mathematical abstraction for the car locomotion, this class keeps all the *"immutable"* physical properties about the car. This allows the car to suffer physical changes in some aspects, and to use different cars, with no changes whatsoever in the rest of the system.

The properties stored in this class are:

- Car Width.
- Car Length.
- Axis distance.
- Turning radius.
- Distance between the rear axle and the front of the car.
- Maximum steering angle allowed by the car steering.
- Maximum curvature allowed by the steering system.
- Maximum velocity.

3.2.2 Car State

The car is a dynamic element in the world. Changing its position around the track at different speeds. Because the car is not a single point in space, to maintain a correct state, the car position needs to include: a point in 2D space and an heading. This is called a **Pose**. However, this pose needs to be mapped in relation to some coordinate system, or else, it would be useless. For now, let's consider that a **World** and a **Section** coordinate systems

exist. In the next chapter, an in-depth analysis of the coordinate systems used in this work will be presented (see 4.5).

Another information that we need to store, related directly with the FNR, is the number of laps that the car as already made. To keep track of this, we use a unique landmark in the track, the Crosswalk section. This section is used as start and end point for the trials, so the number of laps need to be counted here. Because of the shape of the FNR track (8-shaped, see Figure 1.3), in each two lap trial we pass 4 or 5 times through the crosswalk, depending if there is a need to park or not. So, instead of storing the number of laps already made, we choose to keep the half-lap the car is currently in. For instance, before the start of a trial the car is in half-lap 0. When the trial starts the car goes through the crosswalk and the half-lap is incremented to 1.

The car state is implemented by the **CarState** class. This class keeps the car position in the world, and some state variables about the car itself. This information is stored in shared memory space, allowing access to all other modules.

The properties stored in this class are:

- The pose of the car relatively to the world.
- The current section the car is in.
- The pose of the car relatively to that section.
- The last applied curvature for the car - this is derived from the wheel angle.
- The last applied current car velocity.
- The half-lap the car is in.

3.3 Perception and world updating

These modules are responsible for the gathering and processing of sensory data. They are also in charge of updating the World Representation. For this purpose, the ROTA car has multiple perception mechanisms: an odometer sensor, obstacle sensors and 2 cameras.

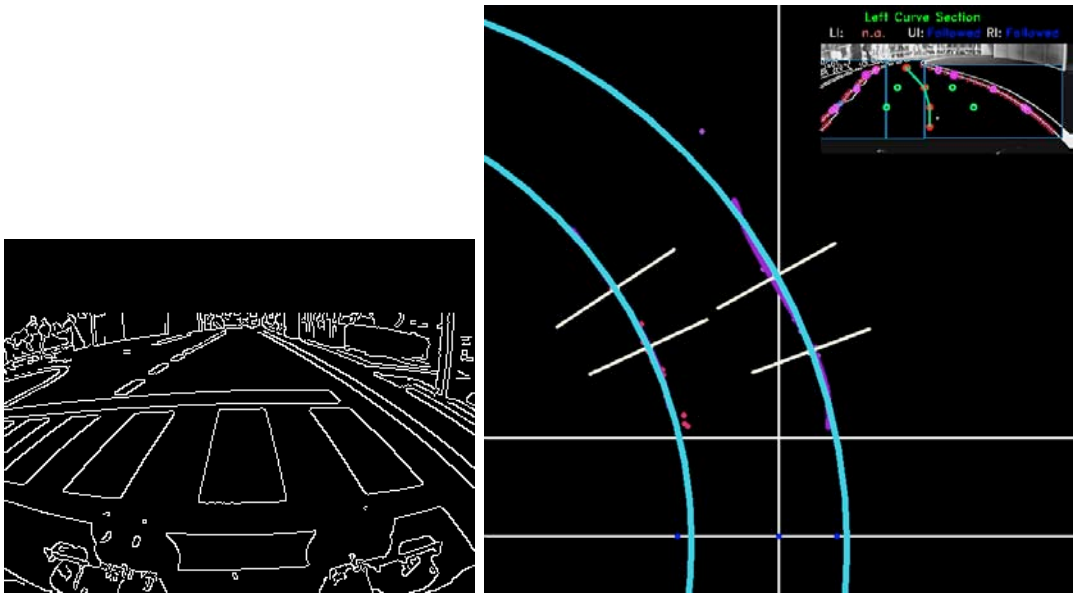
The main components that compose the perception and updating modules are:

- **Road Perception**
- **Traffic Signal Perception**
- **Updater**

The **Road Perception** component is responsible for all of the data processing pertaining with the *road camera*[1]. In this perception mechanism important track elements are located, in order to update the world representation, with not only new track elements, but also updated car positions. Elements that should be located include: the obstacles, the road cones that delimit the roadwork area and the road delimiting lines.

The way this perception mechanism works is as follows:

- After the image capture, the image is processed with a edge detecting algorithm, more specifically the Canny edge detector, see Figure 3.4(a).
- Image points can define important track elements, such as, road delimiting lines, obstacles or road cones, are extracted from the image.
- After being transformed, through an inverse perspective mapping, to real track coordinates, these points are analyzed and classified, in order to extract useful information, such as cone position, obstacle position and lane position (see Figure 3.4(b)).
- This information is passed to the Updater module, which updates the world state accordingly.



(a) Edge detector (b) IPM

Figure 3.4: In (a), the edge detector used in the system is shown. In (b), the road ahead of the robot as perceived by the perception system.

[1]Down-facing camera

The **Traffic Signal Perception** component is responsible for all of the data processing pertaining with the *traffic lights camera*[2]. To identify traffic lights information there are two different libraries: one uses the color information from the image, the other uses template matching.

In the former the dominating color in a specific region of interest is calculated and used to establish which signal is set. However, this technique is dependent on color fidelity. A distortion in the color information of the image, leads to a distorted result. For instance, when direct sunlight hits the traffic lights panels, these appear as a white blob in the camera image, but only from some angles. This is solved by correctly calibrating the camera, to match the light conditions, which can be a time consuming task.

In the second library, a template is created for each possible signal, and then it is compared with the image obtained from the traffic lights camera. Although this method is almost impervious to the light effects seen in the previous method, a different problem affects template matching. In order to correctly establish the signal displayed in the traffic lights, templates from different angles and distances to the panel are needed. This is due to the fact that a change in the car position or orientation, changes the perceived image.

The signals that can appear in the panel, and need to be detected, are depicted in Figure 3.5.

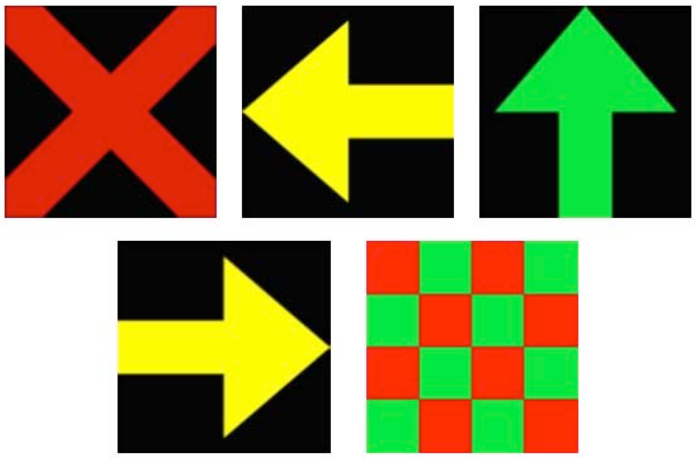


Figure 3.5: Possible signal shown by the traffic lights panels [12].

The **Updater** component is responsible for fusing all the data that the perception components generate. It then uses this integrated data to update the world representation accordingly. In order to improve the update operation, prediction techniques are used to estimate

[2]Up-facing camera

the current car position in the track. For this, it uses the car velocity, odometer information, steering angle and previous track knowledge. These elements are obtained by interfacing with the world representation. The use of prediction techniques together with the perception components output, results in more reliable car positioning in the track.

Additionally to these components, the car model module, which was discussed in the previous section, provides access to either the obstacle sensors and the odometer information. In the current implementation the obstacle sensors are not being used. The odometer information is provided as the distance travelled by the rear wheel.

3.4 Trajectory planning

The **Trajectory Planning** mechanisms are responsible for calculating a valid trajectory, that complies with the car physical properties, car current position, track features and track dimensions, which are stored in the World Representation, but also that respects a given driving strategy.

The driving strategy consists in how the car should progress through the track, *e.g.* driving in the left lane, shortest distance, etc. This driving strategy is then enforced by the use of **Waypoints**, which vary depending on the strategy (if, for instance, we choose to drive in the left lane the calculated waypoints will be, preferably, in the middle of the left lane). A waypoint is defined as the tuple (x, y, θ), where (x, y) is a point in two-dimensional space and θ is a direction (also called heading). A waypoint represents both the point where the car should pass and the heading it should have at that point.

After defining these main waypoints, a trajectory to link them must be obtained, starting from the current pose of the car, and passing through each of the waypoints, taking into account the obstacles in the track, the direction of the car relatively to the section of the track and the physical constraints of the car. These trajectories are defined by means of **WayStates**. A waystate represents a trajectory segment and is defined as a tuple (c, d, v, x, y, θ), where c is the curvature, d the distance to cover, v the desired velocity at the end of the trajectory segment and (x, y, θ) the final waypoint. These waystates are stored in a shared memory space so they can be later used by the path execution mechanism.

As a planned trajectory contains a velocity profile, it is better referred as a motion plan. Figure 3.6 shows a motion plan composed of two waystates.

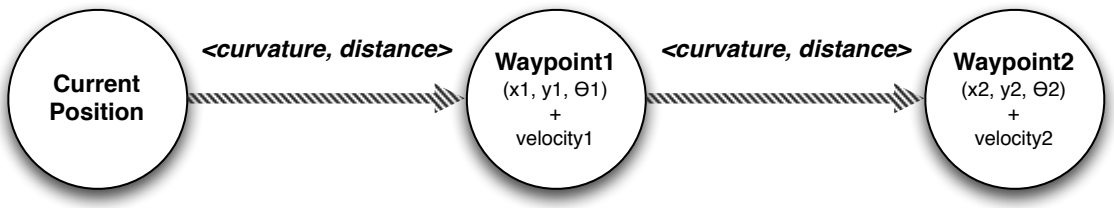


Figure 3.6: Graphical representation of a motion plan.

3.5 Path execution

The **Path execution** is responsible for the execution and monitoring of the motion plan. It is represented by the Driver module, in Figure 3.1. After being calculated by the Planner, the motion plan is now executed. The path execution module is responsible for determining the acceleration needed to achieve the requested final velocity in the waystate. Another responsibility is to keep track of the car's progress in the trajectory, making sure that the car is where it should be. If a considerable deviation from the trajectory is noticed the current plan becomes unsuitable, and needs to be recalculated. As a result the Planner should be notified that a new motion plan is needed.

Besides controlling the execution of the supplied motion plan, the path execution mechanisms can also have a more direct control over the agent's course of action. In case of some unforeseen or unpredictable event occurs in the track this system can have a *final word* on how the agent should behave. Because of this capability to bypass the motion plan that was calculated by the Planner, the ROTA agent can be considered an hybrid agent (see Section 2.1.3).

Chapter 4

Track Representation

Summary

In the previous chapter the overall architecture of the ROTA project software was presented, being the representation of the track identified as a key point. In this chapter a detailed description of the approach used to represent the track is presented. This includes the description of a number of auxiliary functions used for coordinates conversion and perception support.

In order to do trajectory planning both the position of the car and a knowledge of the environment are required. The car position is given by a pose defined in relation to both a general coordinate system and a section coordinate system where the car is at a given moment. The environment is the track and a number of elements that exist around it. The track is considered to be defined in 2D space, since there are no elevations or depressions on it. It is also considered to be composed of several sections connected to each other, each one being a distinctive portion of the track.

The elements that appear in the track are:

- **Traffic lights panel**, which changes the behavior the car should take, defining the direction it needs to take and even stopping it at the crosswalk.
- **Obstacles**, which cause modifications to the default shape of the track.
- **Roadwork area** which, as the obstacles, causes modifications to the default shape of the track.
- **Tunnel** which, because it is a low light zone, can cause problems, mainly in the perception mechanisms.
- **Crosswalk** which, in the FNR competition, represents a major landmark in the track, as it marks both the start and the end of the trial.

Any representation system must also be flexible enough to permit change and access to all of the information about the track and the car in an abstract manner.

4.1 Track sections

A track (see Figure 4.1) can be summarized as being:

A collection of sections which are connected to each other (in a specific way).

Although the above definition is correct, it is incomplete as it does not define what track sections really are, or how to represent the connections between different sections. Lets consider, for now, only the track sections by themselves. The connections between the sections will be discussed further on.

A section is a portion of a track holding some distinctive properties. There are different ways of defining a section, such as:

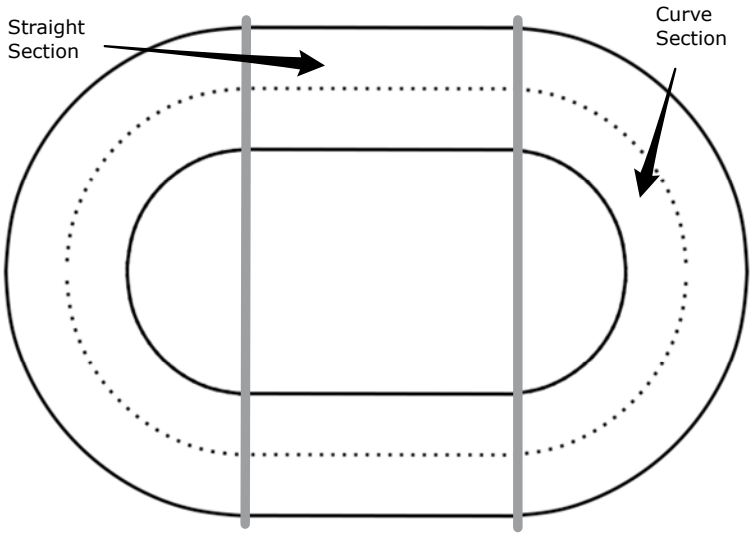


Figure 4.1: A simple oval track, with various sections types.

- A portion of the track between two distinctive landmarks. Considering special points which are unmistakable and unique, a section would be the portion of the track that connects these points together.
- A portion of the track which has a constant shape, *e.g.* a straight portion of the track can be considered a single section.
- A portion of the track which has a constant visual pattern, *e.g.* a portion of the track in which the pattern for the lines, for instance, are constant. In Figure 4.2 there can be seen three different patterns.

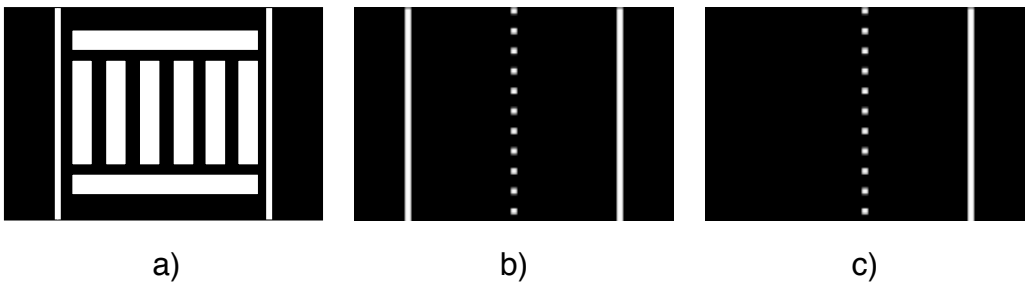


Figure 4.2: Three different track patterns.

The notion of track section used in this project takes into account all these features. Shape and pattern are used to specify what a section is. A section is assumed to have a

given shape and a given pattern. The borders between sections are assumed to have some distinctive feature that allows for a local or global identification. As such, they can be used as landmarks.

Section overlapping

Sections can overlap. This means that a portion of a track can simultaneously belong to two or more sections. This can be seen in Figure 4.3. The curve section and the straight section in front of the crosswalk are partially overlapped.

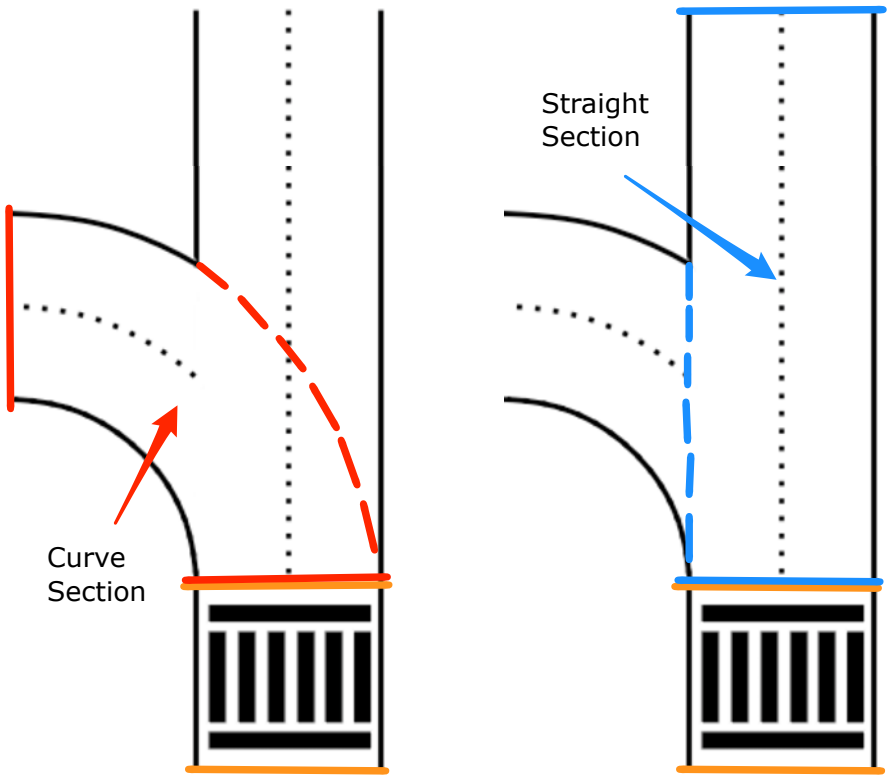


Figure 4.3: Overlapping sections.

Section borders

Sections are connected to each other by means of borders. In the current version of the track model it is assumed a section has only two borders, located at the extremities. So, it is assumed there are no intersections. Figure 4.4 shows a portion of the track split into 3 sections, highlighting the borders between them.

Although in the current version only two borders exist in a section, the model can easily

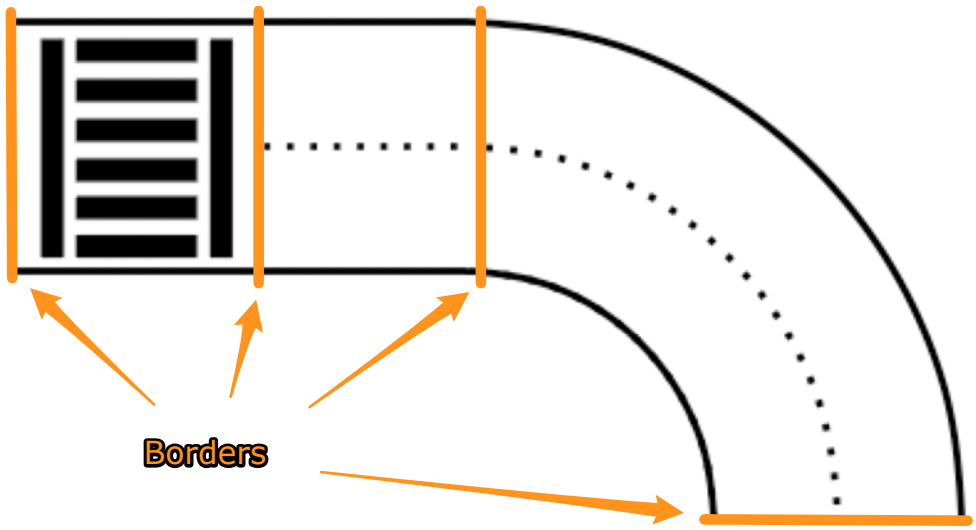


Figure 4.4: A representation of borders.

support borders in other places, different from the extremities of the section. This allows for the existence of intersections in the track.

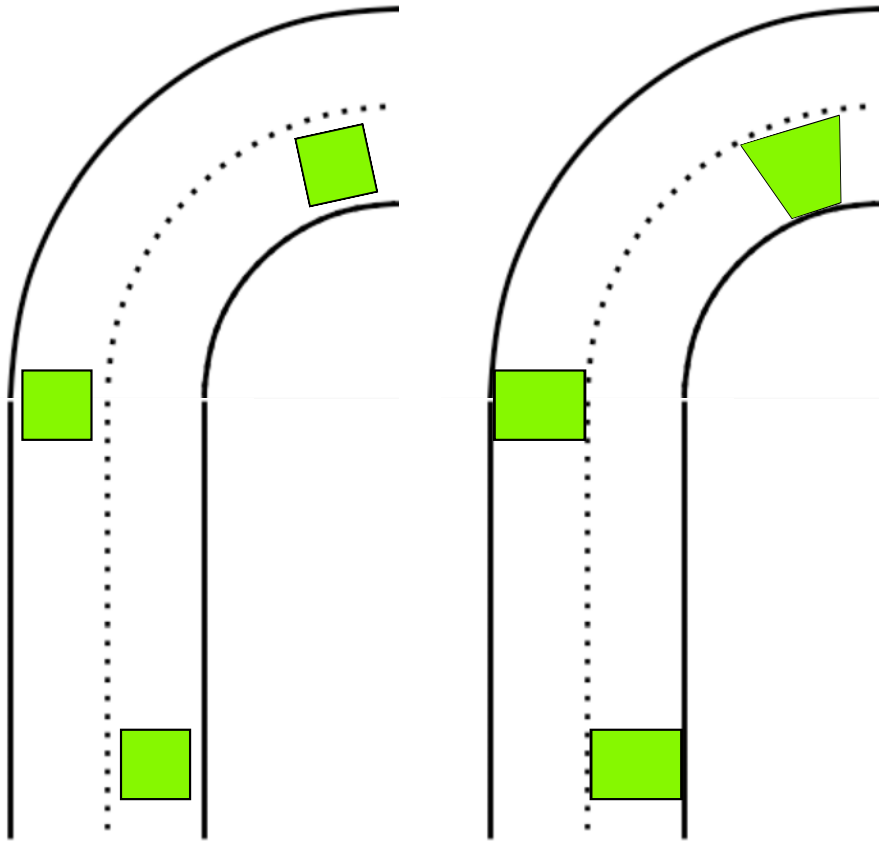
Obstacles

Obstacles are anything in the road that obstruct the driving space. In real scenarios they can be static or dynamic. Examples of static obstacles are a hole or a stone that fell down into the road. Another car, moving in the same lane, is a dynamic obstacle. The obstacles can be anywhere, partially or totally obstructing the road. In the FNR competition, obstacles are assumed to be static and to only partially obstruct the road. They are parallelepiped blocks put over one of the road lanes. So, they turn a segment of a lane unavailable for driving. Figure 4.5(a) shows a track segment and 3 obstacles on it. Obstacle positions are not known in advance, so they are *"dynamic"* to some extent. Because of this, obstacles cannot be used as landmarks.

The model used to represent an obstacle defines it as a range, within a section, where a lane is unavailable for driving. This corresponds to the transversal expansion of the obstacle till the limits of the lanes as illustrated in Figure 4.5(b). An obstacle is represented by the tuple

$$< lane, start, end >$$

where *lane* is the lane and *start* and *end* defines a range within the lane where it is.



(a) Obstacles (b) Transversal expansions

Figure 4.5: Various obstacle positions.

4.2 Section class model

Considering that, at first place, we are preparing to participate in the FNR autonomous driving competition, we start by identifying the type of sections that appear in the FNR track. However the model is open, in the sense that other types of sections can be derived.

Some basic properties had been identified as being shared by all types of track sections. These are:

- a **Name**, a Human readable name, for convenience purposes
- an **ID**, a unique identification number for each section
- a **Type**, a type identifier for the section, *i.e.* straight section, curve section, etc
- a **Width**, the width of the section

- a **Length**, the length of the section
- a **Location**, information about the location of the section in the track, *i.e.* seeing that the world is represented in bidimensional space: X, Y, θ
- a list of the **Obstacles** contained in the section

Some of these properties are illustrated in Figure 4.6.

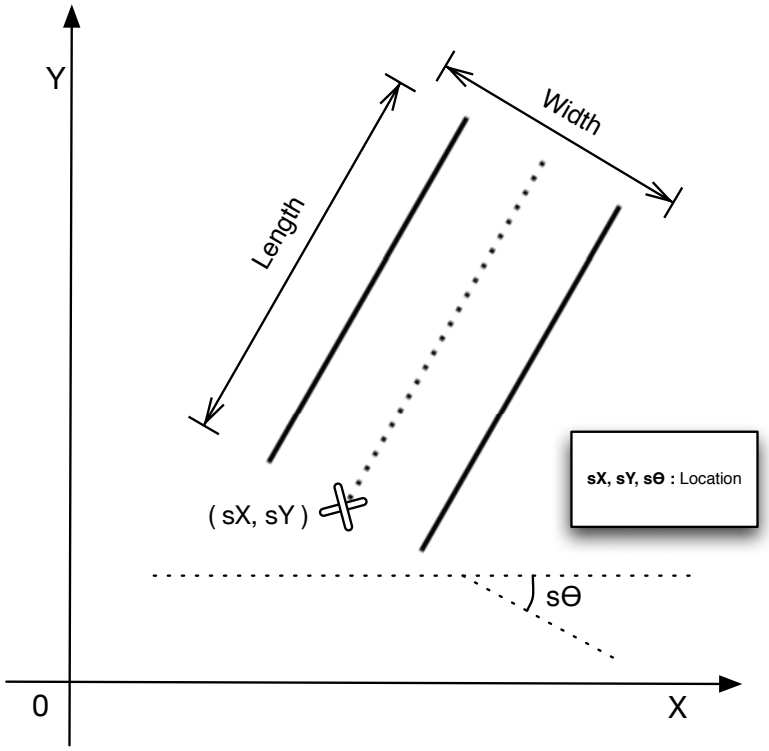


Figure 4.6: Some common track section properties.

Using an Object-Oriented (OO) approach to represent the sections, these common properties can be captured by a base, abstract class. This class, called **Section**, is then used as a base point to derive all the others. Objects from this class cannot be declared. Instead, only objects from a derived class, *i.e.*, from a specific type of section, should be used. This abstract class also declares a collection of functions to support operations that may be performed to or in the sections. Some of these are abstract methods that are only implemented in the derived classes.

Four types of sections were identified for the FNR track. They are the straight, curve, crosswalk and the roadwork area sections, being detailed in the following subsections. Also

an unknown section was introduced to represent the case when the car is facing a road it does not know yet. Figure 4.7 shows the base class, holding the common properties, and the considered derived classes.

To support the storage of ranges, which are used in the obstacles and in the delimiting lines for the sections, a new **Range** support class was defined. This structure represents a normalized range between 0 and 100. This way, if a section only has a center line in the last half the center line property will be defined as

$$[50, 100]$$

which means that the center line starts at 50% of the length for the section, and ends at 100% of its length.

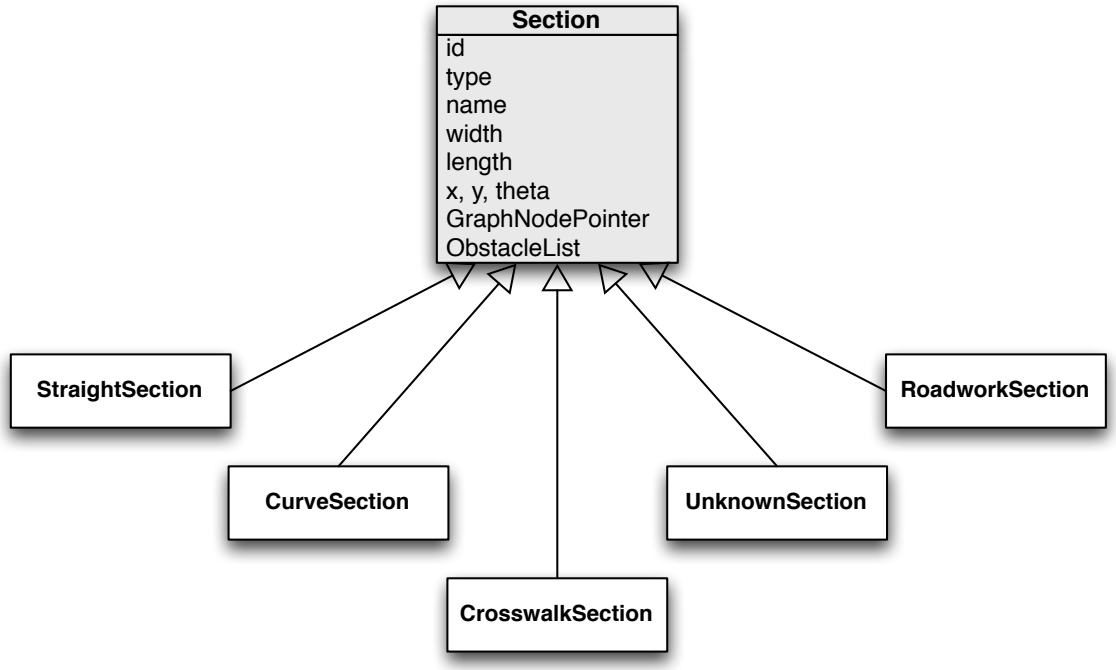


Figure 4.7: Sections class diagram.

Additionally, the section class model, provides high level methods to access or change the section properties. These include add/remove obstacles to a section, check if a given point is contained in a section and methods to convert coordinates (for more information about coordinates conversions, see 4.5), among others.

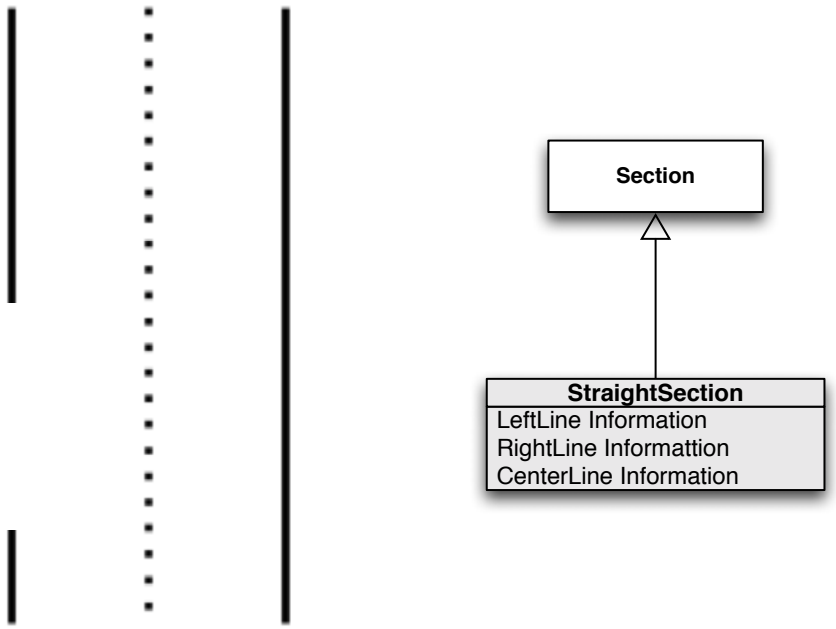


Figure 4.8: Straight section.

4.2.1 Straight section

The straight section represents a portion of the track, which has no curvature, as can be seen in Figure 4.8. This section type is implemented by the **StraightSection** class. The only properties that this type of section adds to the base **Section** class is the information about the delimiting lines. Three lines are considered: a left and a right delimiting line and a central lanes separation line. Normally the formers are continuous and the latter is dashed. But it cannot always be that way. The model allows that any of the delimiting lines can be continuous or dashed. It also allows it to be only partially defined. In order to support that, each delimiting line is defined as a list of ranges, each range defining a delimiting segment. Thus a delimiting line can be partially continuous and dashed.

4.2.2 Curve section

The curve section represents a portion of the track, which has a fixed curvature associated, as can be seen in Figure 4.9. This section type is implemented by the **CurveSection** class. A curve section also has delimiting lines, similarly to the ones defined for the straight section (see Section 4.2.1), but, angular ranges are used instead of linear ones. Also the length for a curve section is stored as the angular distance between the start and the end of the section, instead of the linear distance stored in straight sections.

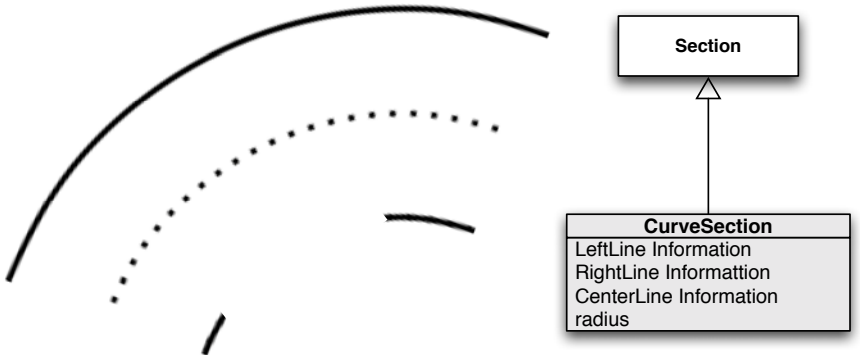


Figure 4.9: Curve section.

4.2.3 Crosswalk section

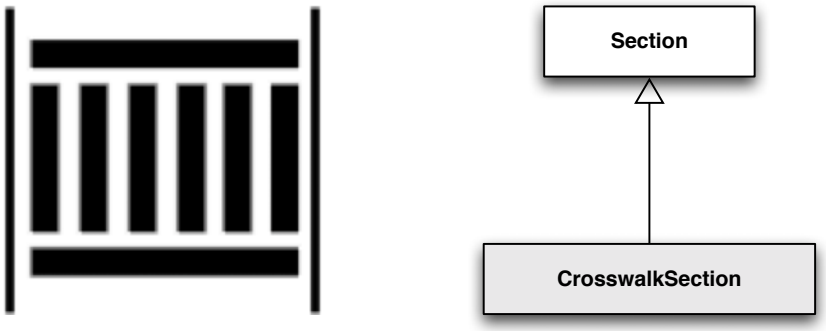


Figure 4.10: Crosswalk section.

The crosswalk section represents a crosswalk in the track, as can be seen in Figure 4.10. This section type is implemented by the **CrosswalkSection** class. This section has some special properties in the FNR competition. It is the start and the end for each of the half-lap around the circuit. Also the traffic light is placed in this section. However, these features are not being associated with the crosswalk section. Instead, they are associated to the track as a whole, this way the traffic lights aren't bound to the crosswalk section.

4.2.4 Roadwork section

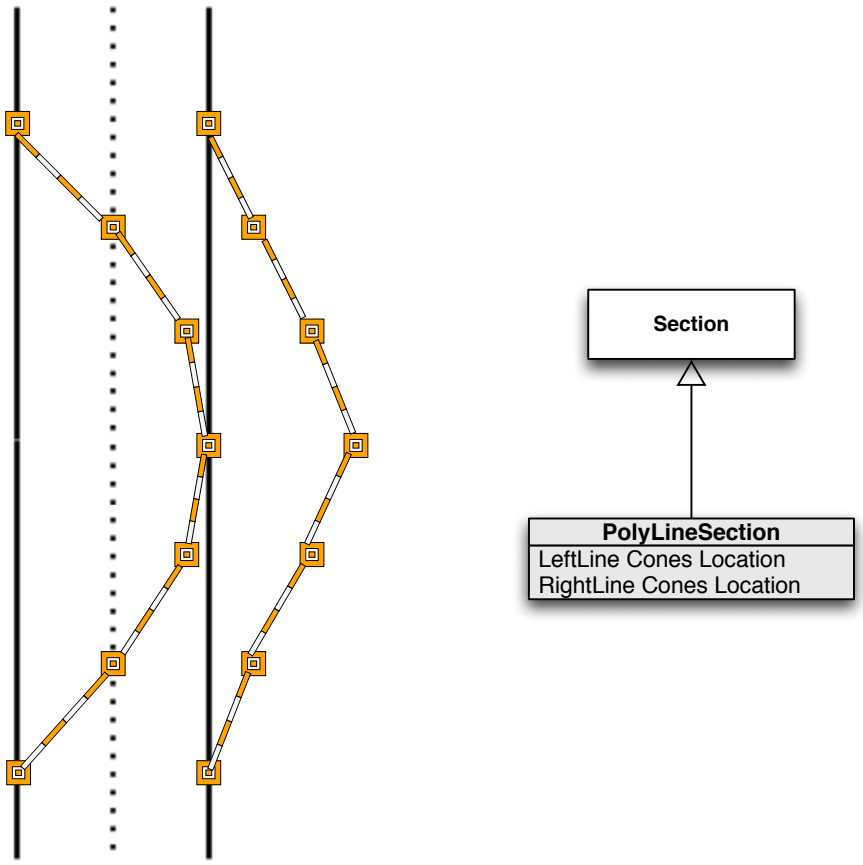


Figure 4.11: Roadwork represented by polyline.

In the last leg of the FNR autonomous driving competition part of the track is in maintenance, defining what we call a roadwork section. This section is unknown in advance. Inside it, the delimiting lines must be ignored, being the driving zone bounded by pairs of orange road cones, connected with red and white strips. The only road cones with a known position are the pairs that start and end the roadwork area. These ones are always placed in the delimiting lines, as depicted in Figure 4.11. This way, we know that a roadwork area always starts and ends in the track. Identification of this zone is only done based on the localization of the cones. So, a roadwork section is defined as a portion of the track without a known shape and laterally delimited by two polylines. The cone positions are used to define the polyline points. The class that defines the roadwork area is the **PolyLineSection** class.

4.2.5 Unknown section

There are some occasions when the car does not know in advance the track morphology. To amend this problem a new section type was introduced: the unknown section. In the FNR autonomous driving competition the first time a car encounters the roadwork area, it does not know how this section will evolve. As such, it does not know what section will be it's successor. Until the car knows which section is the successor to the roadwork area, it creates an unknown section to represent the roadwork next section. Optimally, the unknown section type should only exist temporarily in the model. As the time progresses, and the more knowledge is collected by the sensing mechanisms in the car, these sections should be replaced by a section of one of the other section types. The class that defines the unknown section is the **UnknownSection** class.

4.3 Track model

A **Petri Net** is a directed bipartite graph, which consists of two types of nodes, places and transitions, and directed arcs, connecting these nodes. Directed arcs connect places to transitions, never connecting transitions together or places together. Places are typically represented by circles and transitions by bars. These properties can be better seen in Figure 4.12. Places can contain tokens. These tokens allow the transitions to fire. A transition can fire if all of its predecessors places are marked with tokens. While it fires, a token is removed from each predecessor place and another one is added to each successor place. Tokens are normally represented by dots within the places. In addition to the tokens mechanism, transitions can also have conditions associated with them [18].

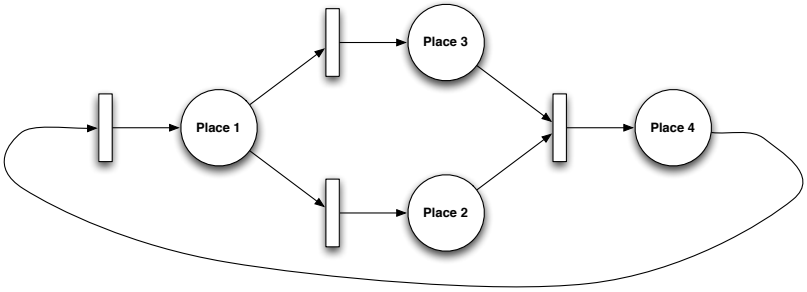


Figure 4.12: A simple Petri net.

A petri net can be used to represent a track, based on sections and borders. A section is represented by a place and a border by a transition. A marking of the petri net composed of a single token, exactly in the place where the car is at a given moment. A transition fires when

the car crosses a border. When this happens the token *"moves"* from the place where it is to the one in *"front"* of the firing transition. This clearly means the car move from a section into another. Figure 4.13 depicts the petri net representation of the simple track (depicted in Figure 4.1, shown before), assuming the car can only move clockwise. Place **P1** to **P4** represent, respectively, the curve on the left, the top straight section, the curve on the right and the bottom straight section. The car is assumed to be in the left curve.

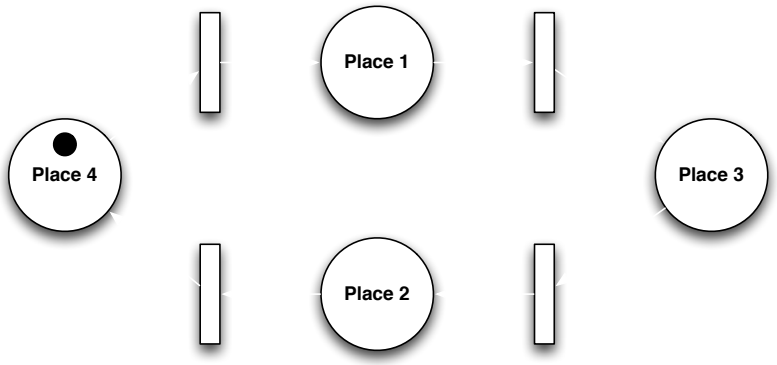


Figure 4.13: Petri net representing track depicted in Figure 4.1.

A condition must be attached to each transition. That condition should be evaluated when the token is at the predecessor place of the associated transition. Only when it evaluates to true should the token be moved to the successor place. Evaluation of the conditions must be done by the updater module, since it involves perception. There could be different ways of doing this evaluation:

- Comparing the section's relative car position with the section length
- Using template matching to identify the crossing of the borders

Transitions store some information: the border of the predecessor section, the border of the successor section and the condition necessary to fire the transition. This means that a transition is represented by the following tuple

$$< b1, b2, cond >$$

where *b1* and *b2* are the borders from, respectively, the predecessor and successor sections and *cond* represents the condition that makes the transition fire.

In Section 2.2 the concepts of topological, metrical and hybrid representations were introduced. The petri net representation corresponds to a hybrid model. At the highest level,

the track has a topological representation, based on the track sections that are connected to each other. On a lower level, inside each of these sections, there is a metrical representation of the section itself.

4.4 Track class model

Tokens represent cars in the track. Since a car cannot be split into two or more cars, nor two or more cars can be merged into a single one, every transition in a petri net representing a track has a unique predecessor place and a unique successor one. Such a petri net can be transformed to a simple directed graph, where places become the nodes and transitions become the arcs. Conditions associated to the transitions must now be associated to the arcs. Figure 4.14 shows a track segment and the corresponding graph representation. The labels of the arcs only represent the borders they connect.

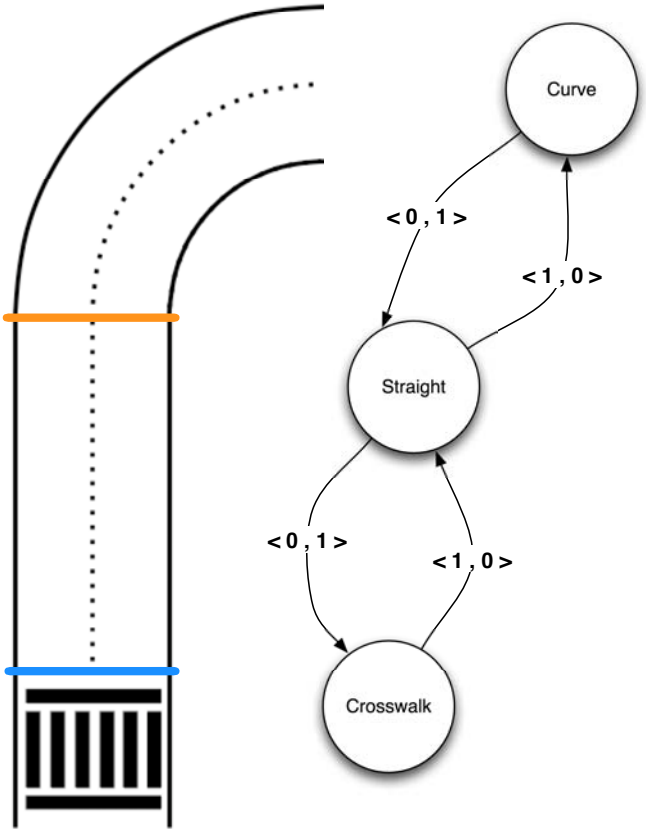


Figure 4.14: A track segment and the corresponding graph representation.

So, although conceptually the track is better described as a petri net, for the implemen-

tation a more widely available graph structure was used. With the information about the transitions being added on the arcs of the graph.

The **BOOST C++ Libraries** are a collection of open source libraries that extend the functionality of C++, aimed at a plethora of C++ users and application domains[19]. Boost include libraries for many purposes, being one of them the **Boost Graph Library (BGL)**[20]. BGL is targeted to graph representation and graph operations, such as graph traversing algorithms. The use of a proven, flexible and reliable graph library such as the BGL, instead of an in-house solution, has a big effect in the diminishing time for the development, and also makes the final solution a lot more impervious to bugs.

Although every node of the graph has an associated track section, their properties are not directly stored in the graph structure. It was decided to separate the section information and properties from the track connection information. The major benefit is that this allows for a more flexible choice in terms of how to support the connection structure and how to store the section properties. For instance, the structure to support the track connection information could be easily changed without any need to change the properties information, and vice-versa.

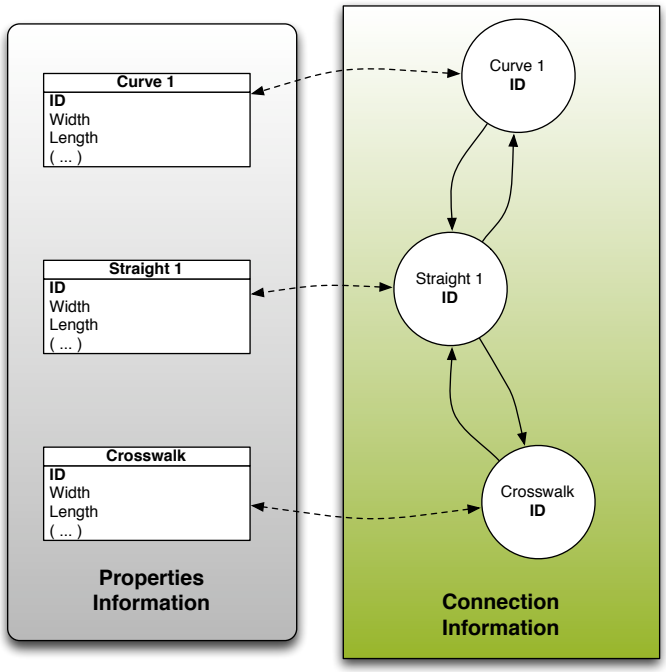


Figure 4.15: Section connections and properties.

Graph nodes and section objects - where section properties are stored - are connected by a double reference. In the graph side, there is a reference (pointer in C++) to the section object.

In the section object, there is a reference to the graph's node. The track segment illustrated in Figure 4.14 has its section objects and graph representation depicted in Figure 4.15.

Additionally, the track class model, provides high level methods to access or change the track properties. These include add/remove sections to a track, add/remove connections between sections, given a point in the world get the section that contains it, among others.

4.5 Coordinate systems

The position of the car can be defined in relation to both a world coordinate system and a section coordinate system. Additionally, there is also a car coordinate system, used to map the environment from the car's perspective. The world coordinate system, also referred to as the global coordinate system, can be arbitrarily defined as any point in the environment. For the FNR autonomous driving competition it was defined with the origin at the center of the crosswalk and with the X and Y axis established as depicted in Figure 4.16.

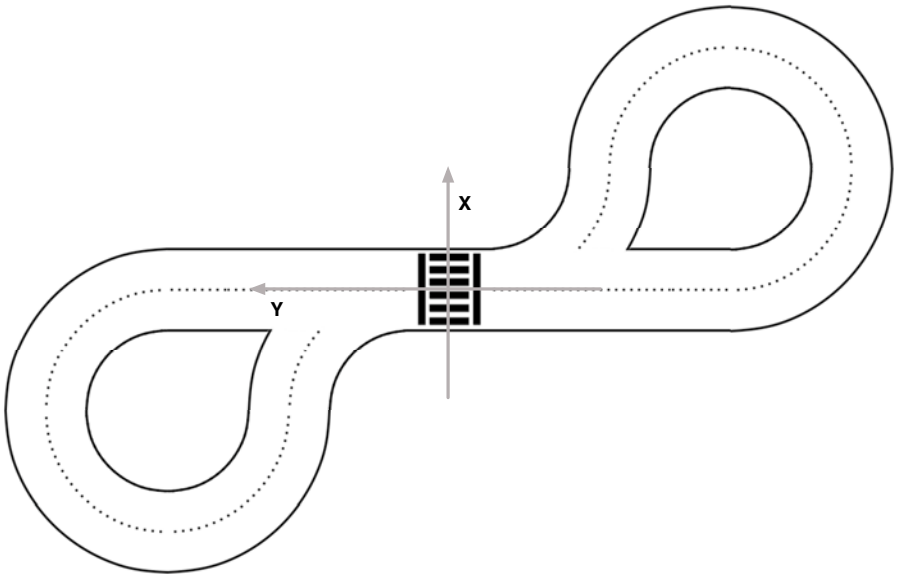


Figure 4.16: World coordinate system for FNR competition.

The car coordinate system has the origin at the center of the rear wheel and the axis established as depicted in Figure 4.17. This coordinate system is used mainly for perception purposes.

Every section has its own coordinate system (see Figure 4.18). Its position and orientation in relation to the world coordinate system is an attribute of the section object.

The type of coordinates used depends on the section type. For the straight and crosswalk

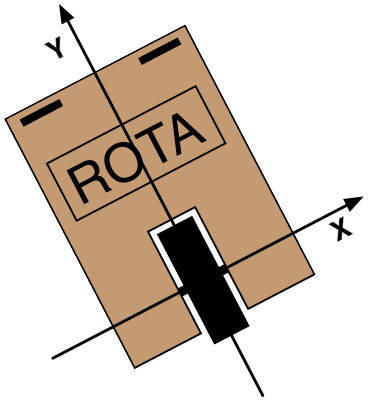


Figure 4.17: Car coordinate system.

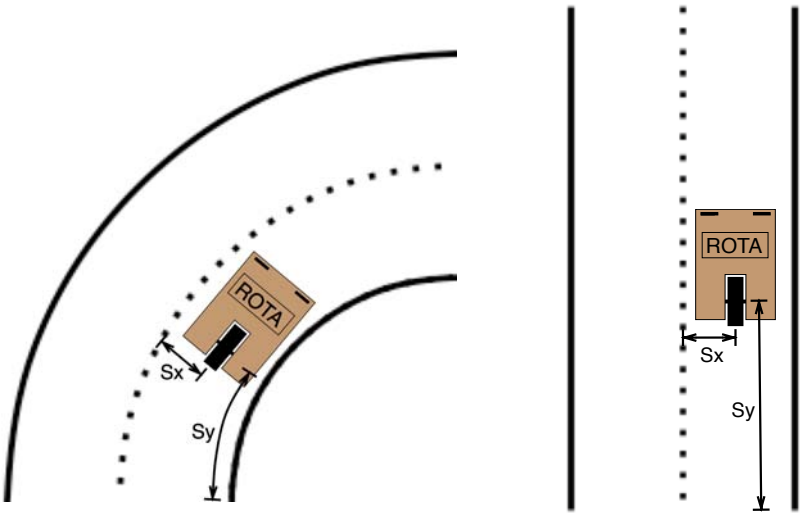


Figure 4.18: Section coordinate system.

section the cartesian coordinate system is used. But for the curved section polar coordinates are used instead. In this case a (Sx, Sy) position represents:

- Sx - the radial distance between the center line of a curved section and the given position.

- Sy - the angle between the start of the section and the radial line that passes through the center of curvature and the given position.

4.5.1 Coordinate conversion

The use of different coordinate systems, brings up the need for reliable conversions between the different systems. Two methods are available to convert from/to section coordinates to/from world coordinates. Because each type of section has different geometrical properties, the type of coordinates varies with it. In order to make the conversion mechanism more flexible, inheritance and method overloading are used. The base class defines two abstract methods, that are only implemented by the derived classes. This is related to the fact that each derived class, being a specific section type, has knowledge about its own geometric properties. So it makes sense that the conversion mechanisms are only implemented by the derived classes.

Notation

In the following conversion formulae the following notation is used.

- $Sx, Sy, S\theta$ - is used to represent a pose in a section coordinate system.
- $Gx, Gy, G\theta$ - is used to represent a pose in the world coordinate system.
- $sX, sY, s\theta$ - is used to represent the location of a given section in the world coordinate system.

Straight section conversions

Equations 4.1 to 4.3 represent a conversion of a world pose to a section pose, in the case of a straight or crosswalk section.

$$Sx = (Gx - sX) * \cos(s\theta) + (Gy - sY) * \sin(s\theta) \tag{4.1}$$

$$Sy = -(Gx - sX) * \sin(s\theta) + (Gy - sY) * \cos(s\theta) \tag{4.2}$$

$$S\theta = G\theta - s\theta \tag{4.3}$$

Equations 4.4 to 4.6 represent a conversion of a section pose to a world pose, in the case of a straight or crosswalk section.

$$Gx = Sx * \cos(s\theta) - Sy * \sin(s\theta) + sX \tag{4.4}$$

$$Gy = Sx * \sin(s\theta) + Sy * \cos(s\theta) + sY \tag{4.5}$$

$$G\theta = S\theta + s\theta \tag{4.6}$$

Curve section conversions

Equations 4.7 to 4.9 represent a conversion of a world pose to a section pose, in the case of a curve section.

$$Sx = \sqrt{(Gx - sX)^2 + (Gy - sY)^2} - |radius| \tag{4.7}$$

$$Sy = \begin{cases} \arctan\left(\dfrac{Gy - sY}{Gx - sX}\right) + \theta, & \text{if } radius \geq 0 \\ -\arctan\left(\dfrac{Gy - sY}{Gx - sX}\right) + \theta, & \text{if } radius < 0 \end{cases} \tag{4.8}$$

$$S\theta = \begin{cases} G\theta - \arctan\left(\dfrac{Gy - sY}{Gx - sX}\right), & \text{if } radius \geq 0 \\ (G\theta - \arctan\left(\dfrac{Gy - sY}{Gx - sX}\right)) + \dfrac{\pi}{2}, & \text{if } radius < 0 \end{cases} \tag{4.9}$$

Equations 4.10 to 4.12 represent a conversion of a section pose to a world pose, in the case of a curve section.

$$Gx = \begin{cases} -(|Sx + radius| * \cos(Sy + s\theta) + sX), & \text{if } \theta \subseteq [\frac{\pi}{2}, \pi] \cap y \geq 0 \\ -(|Sx + radius| * \cos(Sy + s\theta) + sX), & \text{if } \theta \nsubseteq [\frac{\pi}{2}, \pi] \cap y < 0 \\ (|Sx + radius| * \cos(Sy + s\theta) + sX), & \text{otherwise} \end{cases} \tag{4.10}$$

$$Gy = ((Sx + radius) * \sin(Sy + s\theta)) + sY \tag{4.11}$$

$$G\theta = \begin{cases} -(Sy + s\theta) + S\theta, & \text{if } \theta \subsetneq [\frac{\pi}{2}, \pi] \cap (y < 0 \cap radius \geq 0) \\ -(Sy + s\theta) + S\theta, & \text{if} \\ -(Sy + s\theta) + S\theta, & \text{if } \theta \subsetneq [\frac{\pi}{2}, \pi] \cap radius < 0 \\ (Sy + s\theta) + S\theta, & \text{otherwise} \end{cases} \tag{4.12}$$

Chapter 5

Conclusions and future work

Summary

This final chapter gives a global overview of the system and describes the associated work, which is the base for this dissertation. The future investigation and possible expansions are also detailed.

5.1 Conclusions

In this work, an **Architecture** model for an **Autonomous Driving robot with Trajectory Planning** was detailed, with an in-depth description of the track representation model. Until now the ROTA project used an almost reactive-only control scheme. The use of this kind of control made any previous knowledge about the track irrelevant, because it was simply ignored by the robot, when it had to take decisions about the actions to take. This was a serious limitation for the robot performance, in a limited number of situations, during the competitions. With previous knowledge about the track the robot could calculate the fastest route and improve the lap times. Storing the location of already detected obstacles could allow preemptive obstacle avoidance during the trials. This could improve both the lap times and the way the robot handles obstacle related detours.

The lifecycle of the trajectory planning robot can be summarized in the following major steps:

1. The robot starts by collecting information from its surroundings using its sensing mechanisms, such as cameras or proximity sensors.
2. This collected data is processed and the world believes of the robot is updated accordingly.
3. A motion plan is then calculated using this knowledge.
4. Finally this plan is executed.

In order to correctly plan and execute a trajectory, an abstract representation for the track and for the car is needed. Without this representation the planning would be very difficult, not to say impossible. The world representation was developed, with the main purpose of supporting the trajectory planning. For this it can be considered as an abstraction for the track and car properties. It provides an high level Application Programming Interface (API) to the other system modules, in order to isolate the track and car details from the other modules of the system. This representation was designed to be abstract enough, so it can be used in tracks similar to the FNR track, and to support possible changes to the competition track.

The petri net concept is a much better fit than the simple graph for the track connections. The petri net transitions allow for multiple conditions to trigger the change between two sections. This allows for an accurate localization and a more reliable decision making process.

The object-oriented programming (OOP) approach increases the readability of the library structure, in that each object models real world items. The code is easily maintained, and

bugs are easier to find, because different objects are independent. OOP also simplifies the addition of other section types not yet defined or implemented.

Some additional tools were used in the project to support the development. This includes a distributed version control system, in this particular case **GIT**, and an automated documentation generation system, **Doxygen**.

The use of a version control system, such as GIT, allows for an easier code sharing between different developers in the same project. Additionally, in the ROTA project, this system was also used to deploy the code to the robots, and was found to be a flexible and fast way of rapidly test the new changes in the source code.

A source code documentation system, such as Doxygen, that automatically generates documentation, proved to be a very useful tool, easing the integration of the different system components and to allow for a more rapid understanding of the system. For instance, in the world representation module, Doxygen-generated documentation was crucial to other people working in the project, in order to understand and use the external interfaces that the representation module made available.

With the exception of two elements of the world representation, that still are in the initial stages of development (roadwork and unknown sections), all of the proposed goals for the representation system were achieved.

Unfortunately, the architecture model could not be fully tested, and neither could the representation system. Due to the large scale of the changes to the past reactive model of the ROTA system, there was a lack of time to fully integrate all of the components. So no system-wide testing was made.

Nevertheless the representation system was partially tested using a debugging module, created to test insertion, deletion and update of the track elements. It was also tested using the motion planner module, that used the representation to obtain track information and to convert coordinates, in order to calculate the motion plans (trajectories). In these two testing scenarios the system performed as expected, performing changes to the track model and supplying the desired information, not only about the track, but also about the car.

All these elements indicate that both the architecture model and the world representation system, described in this dissertation, are a solid foundation for the coming years in the ROTA project.

5.2 Future Work

During the development of this work some additional tasks were uncovered. These were identified has being possible enhancements to the representation system. Some of these tasks, in no particular order, are:

1. Complete the Roadwork area and Unknown sections implementation, in order to fully support the FNR autonomous driving competition track.

2. Develop a general visualization system to allow for an easier debugging of the representation. Also, this visualization system can have an external interface, in order to allow the other system modules to insert information in the visualization. For instance, the planner module can use this system to display the current motion plan.

3. Although the basic sections found in the FNR autonomous driving competition have been already identified, some additional sections and their properties may be identified and implemented. This could allow the model to become more general purpose.

4. Increase the number of high level API calls. This can simplify the interaction that the other system modules have with the representation system. It can also reduce the probability for error to occur, when doing changes to the model. Consider, for instance, the case when an obstacle is located such that it crosses between two track sections. In this circumstance, the user needs to check in which sections the obstacle is located. After this he needs to calculate how much area of the obstacle is in each of the sections. And only then the user adds the obstacle to the track. However, this includes adding one obstacle to each of those sections. Even assuming the way the obstacle is inserted doesn't change, a better solution would be for the user to add the obstacle to the track, described in global coordinates. And then let the model resolve the object separation between the two sections.

5. Nevertheless the model supports the use of various conditions to trigger the transition between sections, currently the system only uses the difference between the distance traveled inside a section and the section length, as a trigger to the transition. The use of template matching to trigger the transition coupled with the already used condition, can vastly improve the reliability of the localization system.

Bibliography

[1] Wikipedia. Robotics — wikipedia, the free encyclopedia, [http://en.wikipedia.org/w/index.php?title=Robotics&oldid=324781938], 2009. [Online; accessed 9-November-2009].

[2] *Springer Handbook of Robotics, [http://springerlink.com/content/r6m219/]*. Springer Berlin Heidelberg, 2008. doi:10.1007/978-3-540-30301-5.

[3] José Nuno da Silva Carvalho. Rota'2008: um robô para condução autónoma. Master's Thesis, Universidade de Aveiro, 2008.

[4] E.D. Dickmanns, R. Behringer, D. Dickmanns, T. Hildebrandt, M. Maurer, F. Thomanek, J. Schiehlen. The seeing passenger car 'vamors-p'. *Intelligent Vehicles '94 Symposium, Proceedings of the*, pages 68–73, Oct. 1994. doi:10.1109/IVS.1994.639472.

[5] VisLab. Vislab prototypes, [http://vislab.it/Prototypes], 1994-2009. [Online; accessed 21-November-2009].

[6] VisLab. Argo project, [http://www.argo.ce.unipr.it/ARGO/english/index.html], 1997-2001. [Online; accessed 21-November-2009].

[7] VisLab. BRAin-drIVE project (BRAiVE), [http://braive.vislab.it/index.php], 2009. [Online; accessed 21-November-2009].

[8] Robótica 2010. Festival Nacional de Robótica, [http://robotica2010.ipleiria.pt/robotica2010/], 2009. [Online; accessed 8-November-2009].

[9] Robótica 20XX. Festival Nacional de Robótica, [http://www.spr.ua.pt/fnr/], 2009. [Online; accessed 8-November-2009].

[10] José Luís Azevedo, Artur Pereira, Bernardo Cunha, Luís Almeida. ROTA: a Robot for the Autonomous Driving Competition. *Festival Nacional de Robótica 2007 - Encontro Científico*, 2007.

[11] Robótica 2009. Festival Nacional de Robótica, Competição, Condução Autónoma, [http://www.est.ipcb.pt/robotica2009/new/index.php?pagina=competicao], 2009. [Online; accessed 8-November-2009].

[12] Robótica 2009. Rules and Technical Specifications - Autonomous Driving Competition, [http://www.est.ipcb.pt/robotica2009/new/docs/Conducao_Autonoma_2008_12_13_en.pdf]. Technical Report, Sociedade Portuguesa de Robótica, 2008. [Online; accessed 8-November-2009].

[13] Roman Murár. Topological Mobile Robot Environment Representation, [http://dsc.ijs.si/phdworkshop2005/full%20papers/Roman%20Murar.pdf]. *6th International PhD Workshop on Systems and Control*, 2005.

[14] Robin R. Murphy. *Introduction to AI Robotics*. MIT Press, Cambridge, MA, USA, 2000.

[15] Nicola Tomatis, Illah Nourbakhsh, Roland Siegwart. Simultaneous localization and map building: a global topological model with local metric maps. *International Conference on Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ*, Volume 1, 2001.

[16] Scientifica E Tecnologica, Busetta, Bailey J, Ramamohanarao K, Paolo Busetta, James Bailey, Kotagiri Ramamohanarao. A Reliable Computational Model For BDI Agents, 2003.

[17] Nils J. Nilsson. *Artificial Intelligence: A New Synthesis*. Morgan Kaufmann Publishers, Inc., 1998.

[18] Wikipedia. Petri net — Wikipedia, The Free Encyclopedia, [http://en.wikipedia.org/w/index.php?title=Petri_net&oldid=321214842], 2009. [Online; accessed 21-October-2009].

[19] Wikipedia. Boost C++ Libraries — Wikipedia, The Free Encyclopedia, [http://en.wikipedia.org/w/index.php?title=Boost_C%2B%2B_Libraries&oldid=320983142], 2009. [Online; accessed 21-October-2009].

[20] Boost. Boost Graph Library (BGL), [http://www.boost.org/doc/libs/1_40_0/libs/graph/doc/index.html], 2009. [Online; accessed 21-October-2009].